DEPARTMENT OF NATIONAL DEFENCE
CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OPERATIONAL RESEARCH (CORPORATE, AIR & MARITIME)

ORD CONTRACTOR REPORT CR9601

A REVIEW OF SOME DMOR COMPUTER SIMULATIONS

by

Eric L. Leese

DOMUS Software Ltd
309 Cooper Street
Ottawa, Ontario
K2P 0G5

June 1996

OTTAWA, CANADA

National Defence Défense nationale

P499084.PDF [Page: 4 of 97]

OPERATIONAL RESEARCH DIVISION

CATEGORIES OF PUBLICATION

ORD Reports are the most authoritative and most carefully considered publications of the DGOR scientific community. They normally embody the results of major research activities or are significant works of lasting value or provide a comprehensive view on major defence research initiatives. ORD Reports are approved personally by DGOR, and are subject to peer review.

ORD Project Reports record the analysis and results of studies conducted for specific sponsors. This Category is the main vehicle to report completed research to the sponsors and may also describe a significant milestone in ongoing work. They are approved by DGOR and are subject to peer review. They are released initially to sponsors and may, with sponsor approval, be released to other agencies having an interest in the material.

Directorate Research Notes are issued by directorates. They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication. They may record development work done in support of sponsored projects which could be applied elsewhere in the future. As such they help serve as the corporate scientific memory of the directorates.

ORD Journal Reprints provide readily available copies of articles published with DGOR approval, by OR researchers in learned journals, open technical publications, proceedings, etc.

ORD Contractor Reports document research done under contract of DGOR agencies by industrial concerns, universities, consultants, other government departments or agencies, etc. The scientific content is the responsibility of the originator but has been reviewed by the scientific authority for the contract and approved for release by DGOR.

DEPARTMENT OF NATIONAL DEFENCE

CANADA

OPERATIONAL RESEARCH DIVISION

ORD CONTRACTOR REPORT 9601

A REVIEW OF SOME DMOR COMPUTER SIMULATIONS

by

Eric L. Leese

DOMUS Software Ltd
309 Cooper Street
Ottawa, Ontario
K2P 0G5

Recommended by: Pete Smith
A/Team Leader MORT

Approved by:
Director General
Operational Research

ORD Contractor Reports present the results of studies or analyses undertaken by external agencies through contracts for which ORD analysts have acted as the scientific authority. The contents do not necessarily reflect the views of the OR Division or the Canadian Department of National Defence.

OTTAWA, ONTARIO JUNE 1996

P499084.PDF [Page 3 of 97]

ABSTRACT

This Contractor Report examines a number of DMOR computer simulations, already written. Each simulation was tested and, if necessary, corrected to ensure that it worked correctly for various cases of practical importance. Where possible, input variables whose values were 'hard-wired' into the source program were replaced by values that could be assigned at the input stage. Other amendments were made to reduce the execution time of the programs. In one case (ENDLESS), the source code had to be translated from a language suitable for the Macintosh computer to a language that could be compiled by the Microsoft IBM PC software.

RÉSUMÉ

Ce rapport d'entrepreneur examine un nombre des simulations, déjà écrites, de DRO(M). Chaque simulation était testée et, lorsque nécessaire, corrigée pour assurer qu'elle fonctionne correctement pour divers cas d'importance pratique. Si possible, les variables d'entrée dont les valeurs avaient été 'câblées' dans le programme, étaient remplacées par les paramètres qu'on peut définir à l'étape d'entrée. D'autres modifications étaient ajoutées pour reduire le temps d'exécution des programmes. Pour ENDLESS, le code a dû être traduit d'un langage approprié pour l'ordinateur Macintosh, en un langage qu'on peut compiler avec le logiciel Microsoft pour ordinateur personnel IBM.

TABLE OF CONTENTS

Section	Page
INTRODUCTION	1
REVIEW OF SAADS	2
REVIEW OF SEASIM	2
REVIEW OF SHORSIM	3
REVIEW OF ENDLESS	3
REVIEW OF TASMAN	4
REVIEW OF MCS	5
REVIEW OF BPM	5
REVIEW OF SACLANT THREAT MODEL	6
CONCLUSION	7
ANNEX A - REVIEW OF COMPUTER PROGRAM SAADS	8
ANNEX B - SIMULATION SEASIM	19
ANNEX C - SIMULATION SHORSIM	26
ANNEX D - NOTE ON REVIEW OF 'ENDLESS' PROGRAM	31
ANNEX E - NOTE ON TASMAN	41
ANNEX F - PARTIAL REVIEW OF PROGRAM MCS	52
ANNEX G - REVIEW OF PROGRAM BPM	55
ANNEX H - NOTES ON SACLANT THREAT MODEL	59
REFERENCES	83

REVIEW OF SOME DMOR COMPUTER SIMULATIONS

INTRODUCTION

In October 1994, DMOR asked Domus Software Ltd. to carry out a review of a number of computer simulations that DMOR had developed or used. (See ORA Contract Requisition Number W0141-94-SO-12). Each simulation was to be tested for correctness and modified if necessary. Suggestions for improvement were to be developed. The simulations were:

(a) SAADS, a simulation of ship air defence against attacking missiles.

(b) SEASIM, a simulation of ship-borne helicopter operations at sea.

(c) SHORSIM, a simulation of ship-borne helicopter operations on shore.

(d) ENDLESS, a kinematic simulation of the interaction between a ship-launched decoy and an attacking missile.

(e) TASMAN, a simulation of barrier crossing by an escorted convoy, opposed by a single submarine.

(f) MCS, a simulation of minefield clearance at sea.

(g) BPM, a simulation of barrier patrol carried out by a submarine opposing a penetrating enemy submarine.

This review was carried out by Domus Software between 1 October 1994 and 30 September 1995.

Also, in January 1996, under ORA Contract 2207DH-95-AA07, DMOR asked Domus Software to carry out a similar review of another simulation:

(h) The SACLANT Threat Model, a simulation that assesses the effect of varying the amount of ammunition held in reserve at a resupply site, on the effectiveness of a group of ships engaging a number of airborne targets.

This simulation was reviewed between January and June 1996. A brief summary of the work done for each simulation follows. Details are given in the Annexes.

Source and executable code for all simulations (except TASMAN and MCS) were collected on to 1.44 Mb 3.5 inch diskettes. TASMAN resides in the DMOR Shield MicroVax and is stored in directory FITCH. MCS is on a 9-track VAX tape, on a shelf on the South wall of the MicroVax room in the DMOR Shield.

REVIEW OF SAADS

There are three versions of SAADS. The first (SAADS4) was developed by Mr. Omrod of DMOR in 1992. Modified versions PSAADS and BSAADS were produced by Mr. Smith and Dr. Nguyen of DMOR in 1994. All three versions were studied and run. The following modifications to SAADS are suggested:

(a) Prevent program abort, by revising routine INTCPT so that it does not generate an intercept if it would take place outside effective defence weapon range.

(b) Speed up program execution by revising routine INIT to remove the reset of the maximum time step.

(c) Revise the radar detection model to provide more realistic ranges against very low level attackers.

(d) Replace some 'hard-wired' constants by variables that are set at the start of the simulation.

(e) Use integer variables instead of reals for all event times and time steps, using an 0.0001 sec time unit.

(f) Fully document version BSAADS.

Of these modifications (a) and (b) were implemented. Through the SAADS Upgrade Project, Mr. Smith and Dr. Nguyen are implementing (d) and (f) in a new release of SAADS 5.0, to be issued in summer 1995.

Annex A contains a more detailed account of the review of SAADS.

REVIEW OF SEASIM

SEASIM was originally written, in FORTRAN, by Domus Software under a contract to PMO/NSA in 1987. It was extended and modified in 1989 by Mr. Comeau of DMOR.

This version of SEASIM was written for the HP Multics computer then used by DND. The following changes were made to enable SEASIM to run successfully:

(a) Modify SEASIM for the IBM PC.

(b) Build up the COMMON block that was missing from the available SEASIM program disks.

(c) Replace the Multics random number generator by the Microsoft FORTRAN random number generator.

(d) Correct a serious defect in the Microsoft generator, to prevent it from ever generating a random number exactly equal to 0.0. (This caused aborts if one is trying to draw a random number from an exponential, gamma or Weibull distribution).

(e) Revise subroutine STAMP so that it used only four arguments instead of five.

Annex B gives a detailed account of the SEASIM modifications.

REVIEW OF SHORSIM

Like SEASIM, SHORSIM was originally written, in FORTRAN, by Domus Software in 1987 for PMO/NSA, and revised and extended by Mr. Comeau of DMOR in 1989. It was written for the HP Multics system.

The following modifications were made:

(a) Modify SHORSIM for the IBM PC.

(b) Add missing variables to the SHORSIM COMMON block.

(c) As for SEASIM, replace the Multics random number generator by the Microsoft FORTRAN random number generator, and correct the defect in the Microsoft generator.

Annex C gives a detailed account of the review of SHORSIM.

REVIEW OF ENDLESS

ENDLESS was developed by Mr. Hazen of MC/ORD in 1991, and was written in ThinkC Version 3 for the Macintosh computer. It was modified in 1994 by Miss Bajona of MC/ORD, who made the code compatible with ThinkC version 6, and also with ANSI C. She produced 'graphics' and 'non-graphics' versions of ENDLESS. The non-graphics version was designed to be portable to the IBM PC.

Both versions of ENDLESS were installed on a DMOR Mac, but it was necessary to use a Mac with an M68881 coprocessor to make the programs run successfully.

Both versions were moved to an IBM PC, and an attempt was made to run the non-graphics version. The following errors were found:

(a) There was an error in routine bessj1(), used to calculate antenna gain. This caused the program to calculate enormous values of antenna gain that were outside the limits of floating point numbers. Routine bessj1() was replaced by the routine found in the C Volume of 'Numerical Recipes' (Reference (7)). Reasonable values of antenna gain were then produced.

(b) There was an error in the calculation of ship's Radar Cross

Section, leading to a 'floating point divide by zero' abort. The faulty formula was corrected.

(c) Two errors were found in the formula for calculating the minimum miss distance of the missile from the ship. These led to ridiculously large values of miss distance. A better method of calculating miss distance was used. This cured the trouble.

When these errors had been corrected, the 'non-graphics' version of ENDLESS ran on the IBM PC and produced satisfactory output. However, to modify the 'graphics' version so that it would run on the PC, all the graphics routines had to be rewritten in Microsoft C Graphics. The 'graphics' version then ran satisfactorily.

The four modified versions of ENDLESS (graphic and non-graphic, for the Mac and for the IBM PC) were all sent to MC/ORD. MC will now be able to run ENDLESS either on the Mac or on the IBM PC - which is what they wanted.

Annex D gives a detailed account of the changes made to ENDLESS.

REVIEW OF TASMAN

TASMAN was developed by Dr. Craig of the Royal Australian Navy Research Laboratory in 1986, and was made available to TTCP participants. It is written in FORTRAN for the VAX.

The TASMAN program on the DMOR Shield VAX (stored in directory FITCH) was studied and successfuly run. However, study of the source code suggested the following modifications:

(a) Revise the use of the random number generator to produce random normal deviates. Instead of adding twelve uniform random numbers and regarding the sum as a sample from a normal distribution, one uniform random number was generated and converted to a corresponding random normal deviate by using the routine 'ppnd7'. This was a faster and more accurate method of generating a random normal deviate.

(b) In routine 'rtshp', a division operation was replaced by an (equivalent) multiplication. This avoided occasional program aborts due to a 'floating point divide by zero'.

Since a run of the TASMAN program requires the generation of a very large number of random normal deviates, modification (a) resulted in a substantial decrease in running time, of the order of 35%.

Annex E contains a more detailed description of the review of TASMAN.

REVIEW OF MCS

MCS was developed by British Aerospace in the 1980's, and was used by Fleet Technology Ltd. to run several simulations of interest to Canada. Runs were also made by DMOR and by EDRD (formerly DREP).

Attempts were made to locate the MCS source programs in DMOR, but were not at first successful. Dr. Preston of EDRD was asked whether he still had a copy of MCS source, and whether it could be made available to DMOR. In the early 1990's Dr. Preston had worked extensively with MCS, and had found several bugs in the code. He was reluctant to send MCS source to DMOR, since the program was slow and unreliable. He suggested that we get MCS Version 11 from British Aerospace.

Eventually a tape containing the MCS source was found. After considerable difficulty it was found possible to transfer MCS to the DMOR VAX. MCS was then successfully compiled and linked. After a mutually consistent set of input data had been generated, attempts were made to run MCS. The runs always aborted at events that involved classification of a mine contact. Also the output file that gave details of the mine positions, ship movements and events generated, did not make logical sense. In view of the lack of time left in the DMOR Contract, the work on MCS had to be abandoned.

Annex F describes the work done on MCS.

REVIEW OF BPM

BPM was developed between 1982 and 1992 by DMOR staff. The early version was written in FORTRAN for the DND Multics computer system, and assumed that both sides had only passive sonars.

Dr. Yip of DMOR revised BPM to include active sonars, and modified it so that it could be run on an IBM PC. Because BPM used very large amounts of memory, it had to be written in Lahey FORTRAN.

The later version of BPM was transferred to the DMOR Common Area 386 computer, and was successfully run. The individual routines were examined, and the following modifications were made:

(a) The method of generating random normal deviates was improved, as in TASMAN, by generating one random uniform variable and converting it to a random normal deviate by using routine ppnd7. This is faster and more accurate than the current method.

(b) The printout of summary tables of results by routines OUTSTAT1 and OUTSTAT2 was modified to suit the 80-character per line format commonly used by PC printers. (The old BPM output assumed 132 characters per line).

No attempt was made to change the lavish and wasteful use of computer memory by BPM. A few changes (some simple, some not) could reduce the amount of memory used by several hundred KB.

A detailed account of the review of BPM will be found in Annex G.

REVIEW OF SACLANT THREAT MODEL

The SACLANT Threat Model was developed for the U.S. Navy, under contract by QUANTICS. It provides an assessment of the total stockpile of ordnance required, in ships and at a single resupply site, at the start of an operation to ensure that a (specified) 'adequate proportion' of all the attacking targets will be destroyed (or to ensure that some other Measure of Effectiveness will attain a specified value).

For a given weapon type and a given target type, a certain specified number of weapons is assumed to be required to destroy all the targets. All ships have the same initial allowance (IA) of ordnance. When a ship has expended a certain number of weapons (R), it requests another R weapons from the resupply site.

If at the end of the operation it happens that the ships have made more requests for 'resupply chunks' of ordnance than can be handled by the depot, it is assumed that the available 'chunks' of ordnance are distributed at random among the ships.

There is a specified total number of targets, that are distributed among ships in a Homogeneous Polya Distribution. (See Appendix A of Annex H).

If because of supply shortage the total number of rounds fired is less than the specified number needed to destroy the target, the rounds are assumed to be distributed randomly among the targets.

The program works out the initial inventory of rounds required to ensure that the average proportion of ships not running out of ordnance by the end of the operation has some specified value. (In the program supplied by QUANTICS, this proportion is 'hard-wired' to be 0.99).

The present program does not calculate the inventory required to ensure that a specified proportion of the targets are destroyed. QUANTICS memos state that other versions of the program, that work on the bssis of 'expected proportion of targets destroyed', are available. These additional programs have been requested by DMOR.

A very long report, containing several hundred pages of combinatorial mathematics (Reference (19)) derives all the formulae for calculating the results required by the model. However, the report then states that for practical values of number of ships, number of targets, and total number of rounds needed to destroy all the targets, the analytical calculations needed to compute the Measures of Effectiveness are far beyond the capabilities of modern computers.

The report therefore develops a Monte Carlo method of solving the problem. Several thousand replications can be done on an IBM 486 PC in an acceptably short time.

There were some slight troubles in compiling the FORTRAN program as received. When these had been overcome, and the program successfully run, the source code was slightly amended so that it would calculate the inventory needed to produce any desired 'average proportion of ships not running out', rather than the 'hard-wired' 0.99 proportion.

While we were waiting for QUANTICS to respond to DMOR's request to send the other version of the program that calculates inventories needed to ensure some specified average proportion of targets destroyed, the available program was further modified to give results for this Measure of Effectiveness.

At the end of May 1996 QUANTICS finally sent two new versions of the program, that gave much more detailed output. One of these new programs was slightly modified to remove some of the 'hard-wired' features of the QUANTICS program.

A more detailed account of the SACLANT Threat Model is given in Annex H.

CONCLUSION

This Note has described the reviews made of several Naval simulations used by DMOR. The main difficulty was to find the source code and to locate any documentation for the programs. Some errors in the source code were corrected, and some other improvements were also suggested. The reviews were carried out between October 1994 and May 1996.

It is regretted that time was not available for a more extensive review of some of these programs.

Eric L Leese.

(Eric L. Leese)

June 1996

ANNEX A

REVIEW OF COMPUTER PROGRAM SAADS (SHIP AREA AIR DEFENCE SIMULATION)

BRIEF DESCRIPTION OF SAADS

SAADS simulates an engagement between a group of ships equipped with Air Defence Systems, consisting of radars, guns, defensive missiles and chaff, and an attacking force that consists of a number of Anti-Ship Missiles.

Before running the simulation, the user enters data that describe all the relevant characteristics of the defence system components and of the attacking missiles, as well as details of the doctrine that will be used to assign defensive force components to each of the attacking missiles. This is done by responding to a series of 'menu queries' that appear on the computer's terminal screen.

The program determines the times at which each of the attacking missiles will be detected by radar. From this, the program calculates when

(a) tracking radars will have locked on their targets and determined tracks,

(b) illuminating radars are available to illuminate targets as required by the defensive missiles,

(c) defensive missile launchers are ready to launch missiles,

(d) gun fire control systems have obtained a fire control solution,

(e) guns have been loaded, and slewed as required,

(f) defensive missiles have been launched,

(g) a defensive missile has intercepted its target,

(h) gun bursts have been fired,

(i) a gun burst has intercepted its target,

(j) the ship has determined that an intercept was successful,

(k) an attacking missile has started its dive to the ship,

(l) chaff has been launched,

(m) an attacking missile has temporarily changed course because of chaff,

(n) an attacking missile has re-acquired its ship target after being diverted by chaff,

(o) an attacking missile has destroyed its ship target,

(p) an attacking missile has crashed into the sea.

Whenever the overall situation changes - for example, if an attacking missile has been destroyed and the intercept result has been evaluated - the defence weapons are reassigned, according to the selected assignment doctrine, to engage new targets. The simulation continues until either all ships have been destroyed, or all attacking missiles have been destroyed or have crashed.

Every time an attacking missile is intercepted, there are two possible outcomes - the missile can be destroyed or it can survive. We can keep track of the outcomes of all potential intercepts by forming a 'tree' which grows two branches at each intercept time - one for 'attacking missile destroyed' and the other for 'attacking missile survives'. (Other 'intercept' events can also be generated when chaff attempts to decoy an attacker, or when an attacker hits a ship). Each set of possible outcomes corresponds to a 'path' down the tree. Some of these paths lead to a final result 'all missiles destroyed', other paths to 'all ships destroyed', and yet others to intermediate results.

The program calculates the probabilities of each final outcome of the engagement by one of two methods, 'Deterministic' or 'Monte Carlo'.

In the Deterministic Mode, the program runs through all possible paths in the tree (except for those whose probability of occurrence is below a pre-defined small value), and calculates the overall probability of each outcome, then deduces the chance of occurence of outcomes such as 'all ships destroyed', 'all missiles killed', and so on.

In the Monte Carlo Mode, for each replication the program starts at the beginning of the engagement and choses the outcome of each intercept by generating a random number which directs the path down a particular branch of the tree. At the end of the engagement it records and stores the final outcome, then returns to the start to carry out the next Monte Carlo replication. By looking at the incidence of various final outcomes, one gets an estimate of the chance of occurrence of each final outcome.

Naturally the Deterministic Mode produces the more accurate estimates, but for a complex engagement involving several ships and attacking missiles the number of paths down the 'intercept tree' can be extremely large. This of course results in a long program execution time. For example, a 'Deterministic Mode' simulation involving 2 ships, 3 defence systems and 4 attacking missiles had to explore 246 paths down the tree, and took about 30 minutes on a 33 MHz 486 PC. For cases more complex than this it is better to use the Monte Carlo method and accept the reduced accuracy of the results.

At the end of the simulation the program displays the salient results, in summary form, on the terminal screen. It also generates a file which contains all the input data and the simulation results. One can set debug parameters that cause the output file to contain much more detail about the course of the simulation.

PROGRAM DETAILS

The program is written in FORTRAN. It includes 130 subroutines. No special techniques are used.

A programmer's guide, "Ship Area Air Defence Simulation (SAADS) - Programmer's Guide" by F.H.Fitch, CompEngServ Ltd., Nov 92 (Reference (1)) contains complete source code, with partial explanations of each routine, for the 1992 versions of SAADS commonly known as SAADS4. A user's guide to this version of SAADS has been published as "Ship Area Air Defence Simulation (SAADS V 4.0) - User's Guide" by M.K.Ormrod, F.Fitch and M.A.Gammon, Mar 93 (Reference (2)).

Since that time, SAADS has been further developed by Mr. P.Smith and by Dr. B.Nguyen. Mr. Smith restructured the high level routines, some of the deterministic routines, and the statistical output, added improved debugging options and partly documented the common variables. Dr. Nguyen added algorithms and routines to simulate ship motion, wind and the use of seduction chaff. These later versions have not been formally documented, though Dr. Nguyen has written an informal Note that describes, in general terms, how he incorporated chaff into the model. (See Reference (3)). These later versions of SAADS will be referred to as PSAADS and BSAADS.

Since this evaluation in the Fall of 1994, Mr. Smith and Dr. Nguyen embarked on a programme to improve the reliability of SAADS through rewriting, documenting and testing of most of its subroutines, updating the SAADS Users' Guide and rewriting the Programmer's Guide. This upgrade adds a graphical representation utility, ports the simulation to a SUN workstation, and addresses the recommendations of subsequent paragraphs. This upgrade is expected to be complete by the summer of 1996.

[PDF [Page 21 of 67]

TEST RUNS

Versions SAADS4, PSAADS and BSAADS were all run on a PC 486, first using the input given under 'Scenario 1' in Reference (2), and running the Deterministic version of the program.

Version SAADS4 ran to completion in an acceptably short time on the PC 486, and gave results that were identical with those in Reference (2). However when version PSAADS was run with the same inputs, it correctly generated the first 19 paths down the tree and then aborted with an error message, 'intercept out of range'.

Version BSAADS, when run with the same input, aborted in the same fashion. And even after this error had been eliminated (as described below), the program ran for 40 minutes on the 486 before normal completion!

After this trouble had been fixed, BSAADS ran in a reasonably short time (about a minute) and gave results that were identical to those of Reference (2). We then ran BSAADS with seven additional test cases, all of which involved the use of chaff. All these cases ran without trouble, although the last case, which simulated an engagement involving two ships and four attacking missiles, took about 30 minutes to run on the 486.

FATAL PROGRAM DEFICIENCIES

As already mentioned, program versions PSAADS and BSAADS both aborted with an error message when they were tested by running Scenario 1 of Reference (2). This was because an attacking missile that was being engaged by defensive missiles and by guns, had been destroyed by the defensive missile after the guns had fired but before the shells reached the target. This resulted in a 'gun intercept' of the (dead) attacker outside maximum gun range.

To cure this problem, the routine that dealt with intercepts was not allowed to build a new 'intercept node' unless the actual intercept would occur within range of the guns.

A further problem with BSAADS was that, when run with a defensive system that contained no chaff, the initialisation routine reset the 'maximum time increment' for the simulation to zero! Once this had happened, the simulation proceeded in time steps of only 0.01 seconds. Hence the simulation ran extremely slowly.

This problem was overcome by not allowing the offending routine to reset the maximum time step, in a case in which there was no chaff.

P499084.PDF [Page: 22 of 97]

FURTHER PROGRAM DEFICIENCIES

While SAADS ran correctly once the 'fatal errors' had been corrected, there are a few other program features that should be considered for amendment. These are discussed below.

RADAR COVER

At present the program models radar detection by specifying the radar detection range for each of a series of 5 elevation bands (below 5 deg, from 5 to 10 deg, from 10 to 25 deg, from 25 to 50 deg, from 50 to 75 deg). There is a further cut-off below the (optical) horizon.

This model seems far too crude to deal adequately with defence against really low-flying missiles such as EXOCET. For example, if the radar cover is set at 40 km at elevation 5 deg, it will remain 40 km right down to the optical horizon. Surely this is too optimistic.

A better approximation to the radar envelope should be devised, perhaps by using a simple formula for the variation of detection range with target elevation, with a couple of adjustable parameters to shape the detection lobe.

RESET OF MAXIMUM TIME INCREMENT

As explained above, in a 'no chaff' simulation it was necessary to by-pass the resetting of maximum time increment, to avoid very slow program execution.

Discussions with Dr. Nguyen, who modified SAADS to include chaff, suggested that the reset of maximum time increment was not really necessary, even in cases where chaff was used. To test this, the 'maximum time increment reset' portion of the program was completely removed. The program still ran correctly, rather faster than before.

It is suggested that this change is worth making. (Since done).

'HARD-WIRED' VALUES - RADAR ELEVATION PARAMETERS

As stated above, radar range limits are set according to a number of 'elevation bands', whose limits at present are 5, 10, 25, 50 and 75 deg. These values are 'hard-wired' into routine DSPRAD, which is used in getting user input.

However in routine RDSCAN, which calculates radar detection limit, these same elevation parameters are 'hard-wired' into the routine <u>in radians</u>.

If you wished to change these limits, you would have to remember to change the 'degree' values in DSPRAD, and also the corresponding 'radian' values in RDSCAN.

It would be safer to replace these 'hard-wired' constants by variables, whose values could be set initially either by a DATA statement or by assignment statements in routine DEFAULT. Then you could refer to these values, rather than the old 'hard-wired' constants, both in DSPRAD and RDSCAN. (This has since been done by a DATA statement).

'HARD-WIRED' VALUES - INITIAL TIME STEPS FOR BINARY SEARCH

When routine DMEST is determining the 'exact' time at which a defensive missile will intercept an attacker, it sets an initial time step of 3.2 seconds, then repeatedly halves this value until the routine 'zeroes in' on the intercept point. This is done at several places in the routine.

On the other hand when routine GUNEST is looking for the intercept point beteween a gun burst and an attacker, it sets an initial time step multiplier of 4096, which is then halved as many times as is needed to converge on the intercept point.

It is suggested that both these constants be replaced by parameters that are initialised in a DATA statement. (GUNEST and DMEST no longer use this procedure).

'HARD-WIRED' VALUES - DISTANCE TOLERANCES

In some routines such as AMPATH and BEARING, distances are rounded to the nearest 1.0 metres.

This 'distance tolerance' could be defined as a parameter, and initialised in a DATA statement. (Note that the program already sets up a 'time accuracy tolerance' by means of a variable whose value is set in routine DEFAULT).

TIMES AND TIME STEPS AS REAL VARIABLES

Throughout the SAADS routines the time variables that represent events such as detection, firing, intercept and so on, as well as those that represent time steps, are all declared as real variables.

Anyone who has ever worked with FORTRAN (or any other high-level language for that matter) knows the insidious traps that can occur when one tries to compare two real variables, or create a new real variable by adding two reals. And these are the kinds of operation that are repeatedly used in SAADS.

The program tries to get round this difficulty by constantly rounding off the 'real variable' times. But this, besides being a nuisance, does not always cure the problem. In fact one of the SAADS routines contains a patch that was inserted because the program got into an infinite loop.

It is strongly suggested that the program be modified to use integer variables for all such event times and time steps. (This is the usual practice in most of the simulations of which the writer is aware). Then one can safely compare the values of two time variables, or increase an event time by a time step, and know exactly what you have done.

Of course one has to guard against 'integer overflow'. But the SAADS simulation normally runs for at most 500 seconds of real time. Thus if times are expressed in units of 0.0001 seconds, the converted time variables would never exceed 500 X 10,000 = 5,000,000 time units. This is much less than the maximum acceptable value of 2,147,483,647 that is available if 4-byte integers are used (the current standard for most modern computers. To be perfectly safe, declare the variables as INTEGER*4).

To modify SAADS to use integer time variables, one would have to conduct a tedious, almost line-by-line examination and modification of all the SAADS routines. But if carried out correctly, this process would certainly cut out many potential sources of error.

DOCUMENTATION

Version SAADS4 is reasonably well documented in References (1) and (2), though there are some features of Reference (1) that could be improved. For example, the assignment routines could be more fully explained. Routine TSTEP should be clearly explained, so that one can see how the next time step is chosen. Some of the variable definitions are not precise, and some are incomplete or misleading. An overall 'tree diagram' showing how the various routines are generated might be useful, if the inter-relation of about 130 routines can be represented in an understandable fashion.

There are no corresponding documents for PSAADS and BSAADS - just an informal Note that explains in general terms how chaff was added to the SAADS simulation. Full documentation of BSAADS, that explains what all the extra chaff routines do, is badly needed. The documentation could best be carried out by the Defence Scientist who developed the program.

A revised user's guide, covering BSAADS, would be nice but it is not essential.

FUTURE DEVELOPMENTS

SAADS can be regarded as a research tool for 'broad-based' analysis of the defensive capability of a group of ships subject to missile attack, rather than a 'War Game' adjunct that should accept inputs and churn out an assessment as promptly as possible.

On this basis let us consider what should (and what need not) be done to make this program as useful as possible to DMOR.

DISCUSSION

(A) Data Input, now done by responding to 'menu queries' on the screen, is perhaps a little slow but is quite acceptable for a 'research' program. It would hardly be worth changing the input procedure to use the (currently fashionable) Graphic User Interface, using a mouse or the keyboard arrows to select pertinent items. This could of course be done, for example by using Visual Basic to build the input screens, but the improvement in the input task would hardly be justified by the development effort involved.

(B) The subordinate models used by the program (for example, the model of Kill Chance as a function of Range) seem quite adequate for a broad-based simulation, except for the Radar Detection model which, as mentioned above, does not give plausible results for a very low-flying attacker.

(C) The assignment doctrines available in the model seem to be adequate for SAADS purposes.

(D) Program Complexity. The current simulation is certainly quite complex, including about 130 routines. But it is difficult to see how the program could be usefully simplified. The calculation of intercepts, and of defence re-assignments, depends on a large number of factors, and must of necessity be complicated. (For example, routine DMEST that calculates intercepts by defensive missiles, includes 83 sections). (The complexity of the program has since been reduced).

(E) Program Output, both on the screen and in the output file, seems quite adequate for its purpose. In SAADS4 there is a fairly limited debug feature that provides additional output. However in PSAADS (and BSAADS) there is an improved debug facility in which 'debug' can be set to one of 5 levels, separately for each of the 5 major routines. This provides a very good level of output detail.

There might be a case for providing, on demand, a simple (2D) graph of the course of an attacking missile, and of the various defence weapon intercepts. This could readily be done by using the FORTRAN graphics utilities. But it would certainly not be worth graphing the intercepts for all attacking missiles on one diagram. This would be likely to produce an unintelligible plot that resembled the Flight Controller's screen on a busy day at O'Hare.

(F) Program Execution. It has already been noted that it was necessary to make two program alterations - (i) to avoid aborts due to out-of-range intercepts, and (ii) to avoid very slow execution because the maximum time step had been reset. These changes are essential. (These changes have now been made).

(G) SAADS time variables. It has been pointed out that it would be advisable to modify the program so that all time variables are converted to 'integer' values. This change would be tedious and not actually essential, except that it would make the program much safer to use.

(H) Hard-Wired Parameters. As explained above, it would be safer and better to replace the 'hard-wired' constants by variables that can be initialised at the start of the program. These include radar elevation band limits, initial time steps in binary search, and distance tolerances. (Most of these changes have now been made).

(I) Restrictions on the Size of Variables. The program has to include finite limits on the number of ships, of attackers, of defending missiles and guns, and so on. It would be possible to make the program array dimensions dynamic (by using ALLOCATE and DEALLOCATE). But this complication would hardly be worthwhile, in view of the fact that the current program has to keep the array dimensions limited to avoid running out of conventional memory in the execution phase. This space limitation could perhaps be partially removed by calling the various SAADS routines in an Overlay mode. There may be other fixes that remove the conventional memory limitation.

(J) Shuffling of Array Elements. When the defences are being re-assigned after an intercept, there is a lot of shuffling of array elements from one position to another in the event queues. This could perhaps be overcome by using linked lists. It is doubtful whether this complication would be worth it.

(K) Computer Language Used. Nowadays FORTRAN is an unfashionable language, considered by many people to be archaic and obsolescent. However, FORTRAN has a nasty habit of remaining alive and kicking because it is well-known and useful. In addition, recent modifications to FORTRAN make it possible to use several additional statements (such as IMPLICIT NONE, CASE and STRUCTURE) that make the language even more flexible.

It would certainly be possible to re-write the whole program in Pascal or C, or in one of the currently 'in' languages such as C++ or SmallTalk. But the present writer feels that such a lengthy and tricky re-write process would be dangerous, and would not produce any worthwhile improvement.

SUMMARY

To summarise, it is suggested that the following modifications would improve the SAADS program:

(a) Revise routine INTCPT so that it does not generate an intercept node if the intercept would take place outside the effective range of the defence weapon.

(b) Revise routine INIT to remove the reset of the maximum time step, whether the defence forces use chaff or not.

(c) Revise the radar detection model so that it yields more realistic detection ranges against low flying attackers.

(d) Replace the 'hard-wired' constants for radar elevation band limits, for 'binary search' initial time steps, and for distance tolerances, by variables that are set at the start of the program.

(e) Use integer variables instead of real variables for all event times and all time steps, working with time units of, say 0.0001 sec.

(f) Develop complete documentation for the BSAADS version of SAADS. This should include detailed explanations of the DMEST and GUNEST routines, and of the processing of the various queues used in the simulation. It should also explain the rules currently adopted for interference between 'hard' kills by defensive missiles and guns, and 'soft' kills caused by chaff.

(g) (Possibly) add the capability of displaying a graphical presentation of the track of a specified attacker, and the tracks of the intercepting defenders.

Of these, (a) and (b) are essential to make the program run correctly in a reasonable time. (c) would improve the capability of SAADS to estimate defence capability against low-flying attackers. (d) would make the program somewhat more flexible. (e) would make the program safer and more reliable. (f) would of course make the program easier to follow in detail and easier to modify. (g) might illuminate the intercept process, though it might be difficult to implement if the intercepts on a second attacker depended on the results of the intercepts against attacker number one.

P499084.PDF [Page: 28 of 97]

In contrast, the following aspects of the program are considered to be satisfactory as they stand:

(h) The input process.

(i) The assignment doctrine model.

(j) The output data.

(k) Program complexity. This is acceptable because the simulation itself includes so many factors.

(l) The sizes of the program arrays. One must be careful to keep the array sizes moderate, otherwise the program may run out of space.

(m) Use of the FORTRAN language.

CONCLUSION

This Note has given a brief description of the SAADS program, and has suggested some changes that might make the program more useful. It has also listed those features of the program that are satisfactory as they stand.

The revisised SAADS programs are stored on diskettes A, B and C, labelled PSAADS, BSAADS (Part 1) and BSAADS (Part 2).

ANNEX B

SIMULATION SEASIM

INTRODUCTION

In the fall of 1994, Domus Software Ltd. were asked to test and check out the computer simulation SEASIM, which simulates the Anti-Submarine operations of the helicopters on board a Naval Task Group at sea.

The original simulation was written for the Hewlett-Packard Multics computer system, which is no longer in use. To make the SEASIM program useful, it was necessary to translate the FORTRAN routines from Multics format to Microsoft FORTRAN format. This process is discussed in this Note.

BACKGROUND

In late 1987 the Project Management Office for the New Shipborne Aircraft (PMO/NSA) placed a contract with Domus Software Ltd. to develop a computer simulation of the incidence of failures and repairs for the NSA helicopters of a Naval Task Force at sea. The simulation was written in FORTRAN for the DND HP Multics computer system.

In 1989 PMO/NSA asked DMOR to modify and extend the SEASIM simulation, and run it for various cases of interest. Mr. P. Comeau of DMOR worked on this project till mid 1990, when he was posted to STC. A draft report on the SEASIM modifications and the simulation results was prepared, but it was never completed or approved.

SIMULATION ASSUMPTIONS

The simulation studies the incidence of failures and repairs for helicopters attached to a Naval Task Force consisting of one DDH-280 with 2 helicopters, four frigates (CPF's) each carrying one helicopter, and a refueling vessel (AOR) with 1, 2 or 3 helicopters.

The four CPF's are deployed at the corners of a rectangle around the central DDH. The AOR moves from ship to ship on a refuelling cycle. (See Figure 1).

Submarine contacts are (randomly) made by the CPF towed arrays. Helicopters fly out to the contact location, where they drop sonobuoys and attempt to re-detect, locate, attack and destroy the submarine targets. Two helicopters are assigned to investigate each contact. Afterwards each helicopter returns (usually) to its own ship.

P009084.PDF [Page: 30 of 97]

The helicopters can develop snags while in transit to or from the contact, while at the contact region, or while on board ship. Snags are repaired on board ship. For further details about the SEASIM assumptions, see Reference (4).

The DDH helicopters carry out outer screen operations including passive dipping and anti-surface ship targeting (ASST).

During AOR transit the AOR helicopters fly self-protection sorties near the AOR.

SEASIM is an event-driven simulation. 29 event types are distinguished. There are 14 types of time delay between events. Most of these time delays are stochastic. Events are of course processed in chronological order, until the 'maximum simulation time' is reached. Performance statistics are collected for each of a number of replications and averaged. Statistics include

(a) Proportion of planned helicopter sorties completed,

(b) Proportion of submarine contacts intercepted, destroyed or abandoned.

SEASIM RESULTS

Reference (4) presents the results of SEASIM runs made for a large variety of conditions, for a Task Group operation of 6 days. In particular, the Reference shows how the effectiveness of the helicopter Anti-Submarine operations varies with the total number of helicopters carried in the Task Group.

The results of the SEASIM runs were presented to PMO/NSA in July 1989 and April 1990.

MODIFICATION OF SEASIM FOR IBM PC

Since the Multics system is no longer available, it was decided to modify the SEASIM program so that it could be run on the IBM PC.

A floppy disc that contained all the routines for the Multics version of SEASIM was made available by Dr. H. Yip. However, this disc did not contain the "COMMON block" file which held nearly all the variables and arrays used by all routines of SEASIM. The lack of this COMMON file was a considerable nuisance, since the file was INCLUDE'd at the start of the main SEASIM program, and at the start of each subroutine.

P499084.PDF [Page: 31 of 97]

The only solution was to rebuild this file from scratch by a trial-and-error process which involved

(a) Inserting all the variables and arrays mentioned in Reference (4).

(b) Adding any other variables and arrays that were used in any of the SEASIM routines, except for those variables that were used as arguments in the various subroutine calls. (This was done by inspection of the source code for all routines).

(c) Guessing at the dimensions of the various arrays in the COMMON block (maximum number of contacts, ships, helicopters, number of events in the event queue, and so on).

(d) When attempting to compile each routine, as soon as you got an error message that some particular variable was 'used but not defined', you knew that you had to add that variable to the COMMON block.

(e) Compiling each routine by using

```
FL /c /4Yb prog.for
```

By using the /4Yb mode, you ensured that all array bounds were checked during a run. When you got an error message 'Array subscript out of bounds' you were told the number of the source line where the error occurred. By examining the source code you could determine which array was causing trouble, and increase the dimensions of that array in the "COMMON block" file. (This happened several times before the dimensionality of all arrays in the COMMON block was satisfactory).

The final COMMON block file was denoted SEASIM.CMB.

A number of other changes were necessary. One important change was the replacement of the Multics random number generator

```
CALL random_$uniform(IRZ,XX)
```

by the Microsoft FORTRAN random number generator

```
CALL RANDOM(XX)
```

We had already faced this problem when we modified SHORSIM, a Multics program that simulated the operation of shore-based helicopters. (See Annex C). The main difficulty was that the routine RANDOM(XX) generated a value of XX that was greater than or equal to zero and less than 1.0. In a case in which you were trying to simulate a distribution (such as the exponential distribution) in which it is necessary to compute ALOG(XX), the program would abort with an error message as soon as RANDOM(XX) happened to

P499084 PDF [Page: 32 of 97]

generate a value of XX = 0.0. (This actually happened in program SHORSIM).

To avoid this trouble, whenever RANDOM(XX) was invoked in any of the routines, it was immediately followed by

```
IF (XX.LT.1.0E-10)  XX=1.0E-10
```

It was also necessary to change the name of the file LSTAMPX.FOR, which contained subroutine LSTAMP, to LSTAMP.FOR.

The file that contained most of the input variables was SEASIM.DAT. In this file, it was necessary to replace the very large integer given as the initial random number seed by a smaller number, since the large integer was too big for a Microsoft FORTRAN INTEGER*4 value.

A more serious error in the Multics version of SEASIM was that the routine STAMP, which records detailed simulation results in the output file SEASIM.OUT, had five array arguments in the subroutine, and in the calls from the main program SEASIM, but only four arguments when called from routines SEAEVF and SEAEVN. As a result, we got some very surprising and obscure run-time errors! This trouble was overcome by changing the code so that STAMP used only four arguments in all cases.

There were a few other trivial changes in the program.

After the program had been modified in this way, it ran successfully on the IBM 486 in DMOR.

RUNNING SEASIM

After you enter SEASIM, the program asks whether output should go to the screen or to the file SEASIM.OUT. It then asks how much detail should be provided in the output. This can vary from a bare summary of the main results, to a detailed trace of each event in the simulation. (The very detailed response should only be requested when you are doing one iteration. Otherwise your hard drive is going to get filled up in a hurry!)

Next the program asks how many iterations should be done for each case. (100 iterations is usually adequate). Next, the length of the operation in days. (6 days is appropriate).

Next the program asks whether a 'sensitivity analysis' of many cases is required, or whether one case will do. If multiple cases are needed, the program asks for values of

(a) Min, max and increment for Mean Time Between Contacts.

(b) Min, max and increment for Helicopter Availability.

(c) Min, max and increment for Weapons Load.

(d) Min, max and increment for Chance of Kill.

(e) Min, max for number of AOR Helicopters.

The program gets the rest of the (many) input variable values from file SEASIM.DAT. This file can of course be edited if you want to change the value of any of these variables. A list of these variables is given in Table I.

CONCLUSION

This Note has given a brief explanation of program SEASIM, a simulation of the Anti-Submarine operations of the helicopters on board a Naval Task Group at sea.

It also discusses the translation of this FORTRAN program from Hewlett-Packard Multics format to IBM PC Microsoft format.

The revised program is stored on diskette D, labelled 'SEASIM and SHORSIM'.

TABLE 1

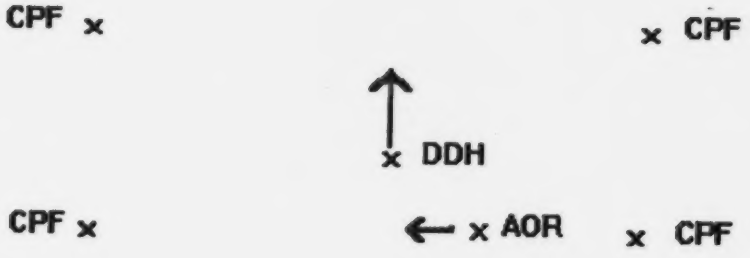
SEASIM INPUT VARIABLES

Maximum number of events in event queue.
Initial random number seed.
Number of ships in task group.
X and Y coordinates of each ship (relative to the DDH) (nm).
Maximum number of helicopters in the Task Group.
Parent ship of each helicopter.
Time unit used in simulation (mins). (Usually one minute).
Planned number of DDH helicopter sorties per day.
Planned take-off time for each DDH sortie (mins after 0000).
Duration of DDH helicopter dipping sortie (mins).
Duration of DDH helicopter anti-surface ship targeting (ASST) sortie (mins).
X and Y coordinates of ASST (nm).
Number of protection sorties per AOR cycle (tour of the Task Group).
Take-off times for AOR protection sorties (mins into cycle).
Duration of AOR protection sortie (mins).
Ships responsible for each protection sortie (see Note below).
Total AOR cycle duration (mins).
Number of kink points in AOR cycle track.
X and Y coordinates of kink points (nm).
Time spent in each AOR track segment (mins).
Convergence zone radii (nm).
CPF first detection probabilities for each convergence zone (%).
Mean time between contacts (hrs).
Mean helicopter down time (mins).
Mean time between failures (in the air)(mins).
Mean time between failures (on deck)(mins).
Percent helicopter operational availability (for factors other than snag repair).
Probability that a snag is "mission critical" (%).
Single weapon kill chance (%).
Helicopter refuel time (mins).
Quick refuel time (mins).
Helicopter take-off delay (mins).
Time until contact abandoned (mins).
Mean time to redetect and localise a target with sonobuoys (mins).
Mean time to attack, given localisation (mins).
Helicopter intercept speed (kts).
Helicopter transit speed (kts).
Helicopter patrol speed (kts).
Task Group speed (kts).
Proportion of DDH helicopter sorties which are Dipping (%).
Number of torpedoes carried by each helicopter.

Note. The ships are numbered arbitrarily. The 4 CPF's are ships 1 to 4, the DDH is 5 and the AOR is 6.

FIGURE 1

SEASIM TASK GROUP



ANNEX C

SHORSIM SIMULATION

INTRODUCTION

This Note gives a brief explanation of the computer simulation SHORSIM, which simulates the incidence of unserviceabilities generated by operational flying and shore-based training for the fleet of NSA helicopters, as well as the processes of snag repair and scheduled maintenance.

BACKGROUND

In late 1987 the Project Management Office for the New Shipborne Aircraft (PMO/NSA) placed a contract with Domus Software Ltd., to develop a computer simulation of the incidence of failures and repairs for the shore-based NSA helicopter fleet. Shore based training flying and scheduled maintenance were included in the simulation. The simulation was written in FORTRAN for the DND HP Multics computer system. Documentation, including source code for the completed simulation, was passed to PMO/NSA in early 1988.

In early 1989 PMO/NSA asked DMOR to modify and extend the SHORSIM simulation. Mr. P. Comeau of DMOR worked on this project till mid 1990, when he was posted to STC. A draft report on the SHORSIM modifications and the simulation results was prepared but was never completed or approved. (See Reference (4)). After this the project lapsed.

In late 1990, at PMO/NSA's request Dr. H. Yip of DMOR started to develop a new simulation to replace SHORSIM. However this work had to be abandoned after a couple of months.

SIMULATION ASSUMPTIONS

It was assumed that the shore-based simulation could be divided into separate operations by East Coast and West Coast forces, with no interaction between the two.

The SHORSIM model included shore-based training and operational flying, as well as support of TASK Group deployments at sea, and scheduled and unscheduled maintenance.

On each coast the pool of available helicopters is supposed to support a planned annual schedule of fleet operations. For each sailing the model specifies the start date, the return date, the number of helicopters required and the planned number of helicopter flying days.

When a sailing is due, the helicopters required are taken from the pool of available helicopters. If this is not possible, helicopters can be "stolen" from the groups of helicopters undergoing minor scheduled maintenance or snag repair. If the ships still cannot get enough helicopters, they go to sea with some helicopters short.

The process of scheduled and unscheduled maintenance, training and supply of helicopters for sea is illustrated in Figure 2.

SHORSIM is an event-driven simulation. 15 event types are distinguished. There are 6 types of time delay between events. Most of these delays are stochastic. The delay types are:

(1) Duration of training or operational sortie (Constant).

(2) Time to repair snag ashore (Random - Exponential).

(3) Time for minor inspection (Random - Distribution provided by sponsor).

(4) Time for major inspection (Random - Distribution provided by sponsor).

(5) Flying time during a sea voyage (Random - Uniform).

(6) Time to repair multiple snags brought ashore (Random - Gamma).

Events are of course processed in chronological order until the 'maximum simulation time' is reached. Performance statistics are collected for each of a number of replications and averaged. Statistics include

(a) the number of helicopters not available for planned sea operations,

(b) the number of training sorties lost.

The assumptions are discussed in more detail in Reference (4).

SHORSIM RESULTS

Reference (4) presents the results of SHORSIM runs made for a large variety of conditions. In particular, the Reference shows how the availability of helicopters for sea duty, and the amount of training flying possible, vary with the total number of helicopters in the fleet, and the policy for scheduled maintenance.

The results of these SHORSIM runs were presented to PMO/NSA in July 1989 and April 1990.

MODIFICATION OF SHORSIM FOR IBM PC

Since the Multics system is no longer available, it was decided to modify the SHORSIM program so that it could run on the IBM PC.

A number of changes were necessary. The set of COMMON variables was put in a file SHORSIM.CMB which was INCLUDEd in each SHORSIM routine. Some of the SHORSIM variables such as QMAX and QMIN, which had been omitted from the list of COMMON variables in the Multics version, were added to SHORSIM.CMB.

A more tiresome change was the replacement of the Multics random number generator

```
CALL random_uniform(IRZ,XX)
```

by the Microsoft FORTRAN 5.1 routine

```
CALL RANDOM(XX)
```

This generator was mainly used in routine SHORKRN, which generates random numbers from uniform, exponential or normal distributions. In several places in SHORKRN the function ALOG(XX) was used. In developing the PC version of SHORSIM, there were several perplexing program aborts with the run-time error message M6202.

This trouble was eventually traced to the use of ALOG(XX) in SHORKRN. Somehow the argument XX had been set to zero.

A study of the Microsft FORTRAN manuals showed that the routine

```
CALL RANDOM(XX)
```

generated a value of XX that was greater than or equal to zero and less than 1.0. In a lengthy run of SHORSIM, it happened that RANDOM(XX) had generated a value of XX equal to zero! Of course when the program then tried to find ALOG(XX), the program aborted.

Routine SHORKRN was therefore patched in several places to avoid the occurrence of a zero value of XX, thus getting round the incredibly stupid behaviour of the Microsoft random number routine. Once this was done, the PC version of SHORSIM ran without error.

SPEED OF EXECUTION

Reference (4) states that, for the Multics version of SHORSIM, it took about 20 hours of mainframe CPU time to run 300 cases of the SHORSIM simulation, for a 3 year operation period and 20 replications of each case. This meant that each case took about 4 minutes of CPU time.

For the PC version of SHORSIM, running on a 486 computer, it took about 36 seconds to run a single case, for 20 replications of a 3 year operation. This suggests that the 486 runs almost 7 times as fast as the Multics, for this particular application.

It would be interesting to put the SHORSIM program on the DMOR SUN computer, to test the running time there.

CONCLUSION

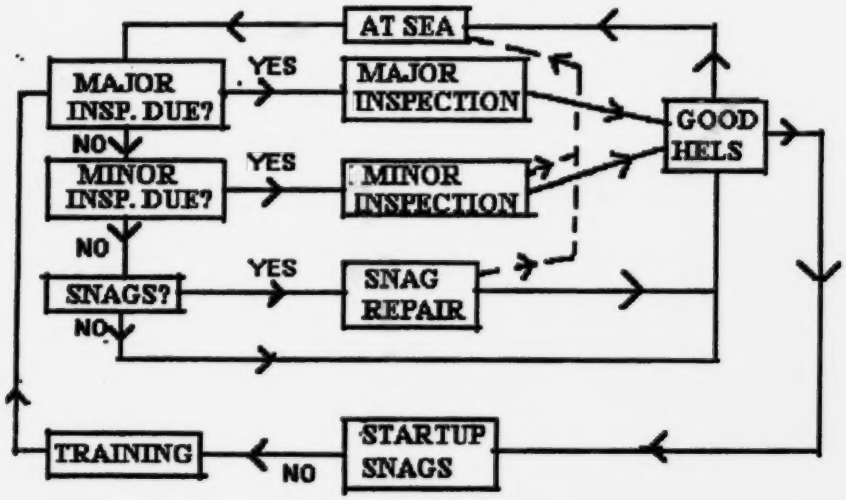
This note has given a brief account of the history and the content of the SHORSIM simulation, and of its modification for the IBM PC.

It is suggested that Reference (4), even though incomplete, should be filed in the appropriate place in the DMOR archives, and that both the Multics and the PC versions of SHORSIM be preserved in a place where they can easily be found.

The revised program is stored on diskette D, labelled 'SEASIM and SHORSIM'.

FIGURE 2

SEASIM OPERATIONS



ANNEX D

NOTE ON REVIEW OF 'ENDLESS' PROGRAM

INTRODUCTION

As requested by DMOR, a review is being made of the simulation ENDLESS, which is a deterministic kinematic model of the effectiveness of decoys (chaff or IR) in preventing an attacking missile from hitting a ship. The minimum miss distance of the missile is calculated for each of a series of conditions (ship course and speed, initial missile range, bearing and speed, and missile and decoy characteristics).

This Note describes the results of test runs of ENDLESS on the Macintosh, and of attempts to transfer ENDLESS to the IBM PC and run it there, with and without graphics display.

BASIC PROGRAM

It turned out that a partial version of the ENDLESS program was available in DMOR, on the Macintosh used by Dr. Mitchell. However this version lacked many of the important ENDLESS routines, and could not be compiled or run.

ENDLESS was originally developed in 1991 by Mr. Hazen of MC/ORD. (Reference (5) gives a good general description of the program). The program was written in ThinkC Version 3 for the Macintosh, and was used at Maritime Command. In the summer of 1994 Miss Bajona, a summer student at MC/ORD, revised ENDLESS so that it could be compiled under ThinkC Version 6. She also revised the code so that it would be compatible with ANSI C, and produced two versions of ENDLESS, one with graphical display and one without. Both versions could be run on the Macintosh, and the non-grahics version was designed to be portable to the IBM PC. (The graphics version contained several 'Macintosh-specific' routines, and was not intended to be portable to IBM-type computers).

Dr. Wheaton of MC/ORD sent a copy of both ENDLESS versions to DMOR.

RUNNING ENDLESS ON THE MACINTOSH

It was found that it was not possible to compile the MC/ORD ENDLESS programs in DMOR, since the version of ThinkC currently available in DMOR was Version 5. Dr. Wheaton therefore sent up some disks containing ThinkC Version 6.

After considerable difficulty, and the transmission of a few more disks from Dr. Wheaton, Mr. Burton of DMOR succeeded in installing ThinkC Version 6 on two of the DMOR Macintosh computers. After this it was possible to compile ENDLESS and run the non-

graphics version at DMOR.

However, when we tried to run ENDLESS with graphics, the program aborted with an error message

'Floating point coprocessor not installed'

This occurred because the MC/ORD ENDLESS program was designed to run on a Macintosh computer that had the M68881 coprocessor, whereas the DMOR Macintosh computers had a RISC chip, and did not need a special floating point coprocessor.

After a few abortive attempts to remove the references to the M68881 coprocessor from the ENDLESS source code, we found that the only solution was to borrow a Macintosh from DLOR that did have the M68881 coprocessor installed. There was then no difficulty in installing the ENDLESS programs on this DLOR computer, and compiling and running both graphics and non-graphics versions.

MOVING 'ENDLESS' TO THE IBM PC

First Mr. Burton used the 'Apple File Exchange' utility to translate the MC/ORD ENDLESS source programs into IBM-readable form. Then these programs were installed on a DMOR IBM 486 PC, in sub-directories \ENDLESS\GRAPH for the graphics version and \ENDLESS\BATCH for the non-graphics version. (Incidentally the names of the ENDLESS source files had to be truncated to suit the requirements of DOS).

The next step was to try and compile and run the non-graphics version of ENDLESS on the IBM PC.

ERRORS IN ENDLESS (NON-GRAPHICS VERSION)

The ENDLESS source files were successfully compiled on the IBM 486 by using

CL /AL /c /FPi87 (program).c

After linking, we attempted to run the program. Some errors were found. Some caused the program to abort. Others gave ridiculous output results. These errors will be discussed in turn.

FLOATING POINT OVERFLOW

The first run aborted with an error message

'error M6104:MATH - floating point error: overflow'

Since Microsoft C does not seem to provide any easy way of determining just where a run-time error occurs, it was necessary to put in a lot of extra 'printf' statements in the program to narrow

down the source of the error. It was eventually traced to a mistake in the MC/ORD function bessj1() that calculates an approximation to the Bessel function $J_1(x)$. In the function antenna_pat(phi) which forms part of file decoy.c we tested the statement

```
gain = 2 * bessj1(epsilon)/epsilon;
```

by printing out the values of 'epsilon' and 'gain' at the point at which the program abort was about to occur. We found the values to be

```
epsilon = 0.160221

gain    = 29385777788573730000000000000000000000.0
```

Since the next statement in the program raised 'gain' to the fourth power, it was little wonder that a floating point overflow error occurred.

We found that the Macintosh bessj1() utility had been developed by taking the appropriate Bessel function program from the FORTRAN version of 'Numerical Recipes' (Reference (6)), and translating the FORTRAN statements into C. We tried the effect of replacing this bessj1() utility by the program given in the C version of 'Numerical Recipes' (Reference (7)). When we recompiled ENDLESS and tried to run it again, this particular error disappeared.

FLOATING POINT DIVIDE BY ZERO

However, the ENDLESS run now aborted at another place, with a run-time error message

'M6101:MATH floating point error: invalid'

To trace where this vague and unhelpful message had arisen, once again we inserted a lot of extra 'printf's in the program. We finally narrowed down the offending statement to

```
rcs_angle =

(view_angle/fabs(view_angle))*(fabs(view_angle)+0.04363);
```

We ran the program, printing out the values of view_angle and fabs(view_angle), and found that

```
view_angle = fabs(view_angle) = 0.0
```

Thus the first part of the offending statement divides 0.0 by 0.0, creating a floating point divide by zero, and aborting the run.

We found that this portion of the program was calculating the Radar Cross Section of the ship for the given value of view_angle, the direction of the missile relative to the ship's bow. This is done by using an array RCS[] that gives the Radar Cross Section for each 5 degrees of 'view_angle' from - 180° to + 180°. We have to replace 'view_angle' by the nearest angle that is an integral multiple of 5 degrees, and this angle is mapped into an integer between - 36 and + 36. Because of the way that computers handle the truncation of negative real numbers, (for example, replacing -4.7 by -4), one needs to use a trick to get the computer to replace a real number 'n (-36.0,36.0) by the closest integer. And the ENDLESS program happened to use a method that resulted in a floating point divide by zero.

Since we know that x is always greater than or equal to -36.0, another way of doing the job is to replace x by

$$y = x + 36.5$$

then truncate y to an integer and subtract 36 from the result.

We used this method in ENDLESS by replacing the troublesome statement, and the following one, by

```
jj=(int)((RAD_DEG(view_angle)+182.5)/5.0)-36;

ts=antenna_pat(phi_s)*RCS[jj];
```

in the case of a chaff decoy, and

```
jj=(int)((RAD_DEG(view_angle)+182.5)/5.0)-36;

temp_s=IR[jj];
```

in the case of an IR decoy.

These changes eliminated the problem of the 'floating point divide by zero'. The ENDLESS program now ran to completion and we began to get some results. However, these results were sometimes ridiculous.

ERRORS IN CALCULATION OF MINIMUM MISS DISTANCE

We used the default ENDLESS input data, and ran the program for the following cases:

(a) Ship course from 0° to 90° from true North, with

(b) Missile initial bearing from 0° to 90° from true North.

The output file data.out gave a table of minimum miss distance for each of the various cases. For most cases the distance looked

reasonable (less than 300 yds), but for the case of ship's course = 30° and missile initial bearing = 0°, the 'minimum distance' was given as - 43510.1 yds!

We ran the same case on the Macintosh. This time the minimum miss distance was given as - 99340384.0 yds!

By chance we had two sample 'input data' files for ENDLESS. When we ran ENDLESS using the first file, we got the above silly results. But when we used the second input file we got sensible results in all cases.

We compared the two input data files. They were identical, except for the range below which the missile seeker was disabled. This range was 500 yds for the first file, but 250 yds for the second file.

A look at the ENDLESS function seduction() which computed the missile track, showed that the 'missile to ship distance' was calculated for each time step. As soon as this distance started to increase, it was assumed that the missile had passed the ship. If the missile seeker was enabled at the time at which the 'missile to ship distance' was smallest, that distance was recorded in the output file as 'minimum miss distance'.

But if the missile seeker was disabled at this time, a correction 'R_hull' was subtracted from 'minimum miss distance' before the value was passed to the output file. The value 'R_hull' was supposed to be the distance from the ship centre to the edge of the ship, in the direction towards the missile.

By printing out the values of 'minimum miss distance' and 'R_hull' when we ran ENDLESS, we found that according to the (IBM 486) program:

minimum miss distance = 131 yd, but R_hull = 43641 yd!

Hence the 'minimum miss distance' passed to the output file was

131 - 43641 = - 43510 yd!

A closer look at the ENDLESS seduction() function showed that

(a) The calculation of 'missile bearing relative to ship's bow' was wrong, and

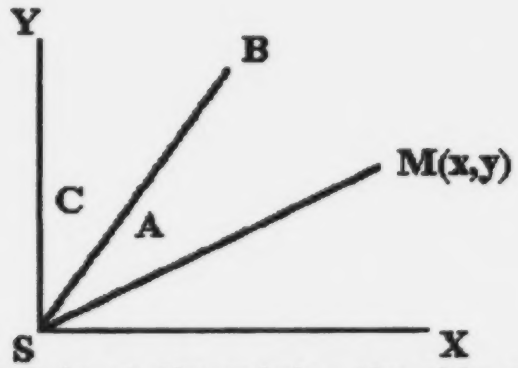
(b) The function that calculates R_hull contained a couple of mistakes.

CALCULATION OF MISSILE BEARING FROM SHIP'S BOW

In function seduction(), the missile bearing A_Angle is calculated as follows:

```
x_del = x_m - xs;
y_del = y_m - ys;
x_sh = cos(s_course);
y_sh = sin(s_course);
temp_Angle =
((x_del*x_sh)+(y_del*y_sh))/sqrt(_sqr(x_del)+_sqr(y_del));
A_Angle = acos(temp_Angle);
```

Here (x_m,y_m) is the missile position, (xs,ys) is the ship position and s_course is the ship's course relative to true North. The problem can be represented by the following diagram:



Here the ship S is at the origin, the ship course SB is at an angle C from true North SY, the missile is at M with coordinates (x,y) relative to the ship, and the missile bearing is at angle A off the ship's bow.

The above lines of code attempt to calculate angle A by getting the components of unit vectors along SB and SM, and then taking the dot product of the two vectors to get cos A. Unfortunately, the program assumed that the unit vector along SB had x- and y-components (cos C, sin C). This is wrong - the components are in fact (sin C, cos C). Hence the calculated value of angle A is wrong. Moreover there is an ambiguity in deriving A from cos A. It is much simpler and less dangerous to calculate

$$\tan (C + A) = y / x$$

using the atan2 function, and then subtracting the known angle C to get A.

These errors are somewhat irrelevant since, as we will see

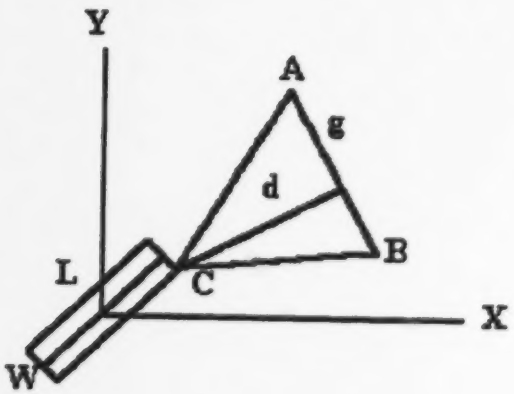
shortly, there is a much more direct and better way of calculating minimum miss distance.

ERRORS IN CALCULATING R_HULL

There is a typing error in the first 'if' statement in function hull_range - the last minus sign should be a 'plus'. This mistake pushed the calculation of R_hull into the wrong section of the routine (the 'else' section). Since in any case the wrong value of argument AA was passed to the routine, the value of A_comp happened to be very close to 90°, so that tan(A_comp) was very large. This led to the enormous value of R_hull reported above.

A MORE DIRECT METHOD OF CALCULATING MINIMUM MISS DISTANCE

Rather than correcting the above errors, it was considered that a better estimate of minimum miss distance could be obtained by computing ship and missile positions at each time step, but 'remembering' the missile position relative to the ship at the previous time step. Then, when the missile to ship distance started to increase, you could do linear interpolation, using the 'missile position relative to ship' at the current time step, and at the previous time step, to determine the minimum distance from the missile to the edge of the ship (regarded as a rectangle). The situation is as shown in the following diagram:



Here the ship S is at the origin, and B and A are missile positions at the current time step and at the previous time step. C is one of the 'corners' of the ship. The coordinates of A, B and C are

(x_a, y_a), (x_b, y_b) and (x_c, y_c).

Let g = distance from A to B, and d = perpendicular distance from C to AB. Then the area of triangle CBA is dg/2. But this area is also D/2, where D is the determinant

$$D = \begin{vmatrix} x_c & y_c & 1 \\ x_b & y_b & 1 \\ x_a & y_a & 1 \end{vmatrix}$$

Hence $$dg = x_c(y_b-y_a) + y_c(x_a-x_b) + (x_by_a-x_ay_b)$$

and $$g = \text{sqrt}((x_b-x_a)^2 + (y_b-y_a)^2)$$

Thus we can easily calculate the value of d for each of the four 'corners of the rectangular ship'. The sign of d depends on whether the point C is to the right or to the left of the missile track AB.

If all four d's are positive, the minimum missile miss distance is the smallest of the d's. If all four d's are negative, the miss distance is minus (the largest of the d's). If some d's are positive and some negative, the miss distance is zero.

We developed a small function min_dist() to compute the minimum distance in this way. In addition, we made a few changes to function seduction() to use the above method of calculation.

When we recompiled and ran ENDLESS, we found that the values of minimum miss distance seemed plausible for all the cases we tried.

We also made these changes in the Macintosh version of the non-graphics ENDLESS program. The program then gave acceptable results.

The next step was to modify the Macintosh Graphics version of ENDLESS so that it could be run on the PC.

PC VERSION OF ENDLESS (GRAPHICS)

It is not always easy to do a one-for-one translation of the Macintosh ThinkC graphics routines into corresponding Microsoft C graphics routines. It was therefore decided to start from the non-graphics PC version of ENDLESS, discussed above, and to add Microsoft C code that would (fairly closely) produce the same screen output as that of the Macintosh Graphics version.

This job fell into two parts. The first was to write the code that would enable the user to modify the default input set of variables contained in file input.def, and to set up the type of plot required. The second was to rewrite the plot routines.

MODIFYING THE DEFAULT INPUT VARIABLES

This was mostly straightforward, except that it was necessary to rewrite the ipread() and fpread() functions used extensively in the process of input parameter modification. (Straightforward translation of these two functions from ThinkC into Microsoft C did not seem to work).

We set up these functions so that you could use 'TAB' or 'ENTER' to move from one input parameter to the next. We made no attempt to emulate the Macintosh version in which you could use the mouse to reposition the cursor just in front of any specified input parameter. This would have been possible, but it would have taken a lot of time.

Also in analysing the keyboard responses to screen queries, we found it necessary to make heavy use of getch() and getche(), rather than getchar(), to get the desired 'unbuffered' response.

To avoid replacing a very large number of printf() statements by sprintf() plus _outtext(), we used a white-on-black screen for the input modification section of the program. When it came to the plots, we used black-on-white (the 'Mac' standard).

In input modification we used a video mode of _TEXTC80, since the cursor remains visible only in text modes.

We did not bother to change the font for the first page of screen output (the heading "The Effective Naval Decoy Launch Envelopes for Single Ships Model"), since to use registerfonts() you need to know the pathname of the directory that contains the *.FON font data - and this pathname could have been machine-specific.

MODIFYING THE PLOT ROUTINES

In this part of the program we set the video mode to _VRES16COLOR.

The plot routines that drew the ship track, the missile track, the flare tracks and the 'centre of apparent target' positions, were quite easy to modify.

In the case of the 'chaff rectangle' plots, it was necessary to erase the 'old' chaff rectangles before you drew the current rectangle. We could not adapt the 'erase' technique used in ThinkC to Microsoft C language. Instead we erased the rectangles by temporarily changing the foreground colour index to WHITE, and then redrawing the old rectangle.

This worked well except that when the old rectangle overlapped part of the ship's track, that part of the ship's track got erased along with the old chaff rectangle. We could see no easy way of curing this. Still, perhaps our method isn't much worse than the ThinkC method, which leaves a dribble of lines from the corners of the chaff rectangle, as the rectangle advances down the screen.

As in the case of the Macintosh plots, our flare plots show the flares to be fat lines proceeding down the screen. It would be quite easy to wipe out the old 'flare crosses' before the new ones are drawn, thus representing the flares as a set of moving crosses.

CONCLUSION

This note has discussed the Graphics and non-Graphics versions of ENDLESS, has pointed out some errors in the non-graphics version and has shown how they can be corrected.

It has also discussed the conversion of the Macintosh ThinkC ENDLESS programs into a form suitable for running on the IBM PC. These PC versions have been tested, and appear to run satisfactorily.

Floppy discs containing the amended versions of the Macintosh ENDLESS Graphics and non-Graphics programs, and of the IBM PC versions of both programs, have been sent to MC/ORD. MC/ORD have since tested these four programs, and they run correctly. Copies of the programs are stored locally on diskettes E and F, labelled ENDLESS, for IBM PC and Mac.

ANNEX E

NOTE ON TASMAN

INTRODUCTION

In 1986 Dr. Craig of the Royal Australian Navy Research Laboratory developed TASMAN, a computer simulation of a barrier crossing by an escorted convoy, opposed by a single submarine. TASMAN was developed for TTCP, and was written in FORTRAN for the VAX.

Each of 3 escort vessels is equipped with a towed passive sonar array, and a single helicopter. When an escort obtains a submarine contact it launches the helicopter, which flies to the predicted target location and drops sonobuoys to re-detect, localise, track and engage the submarine. The submarine is equipped with a towed passive sonar array.

The convoy is attempting to cross the barrier and reach a 'goal'. Each vessel (convoy, escorts and submarine) may take evasive action according to a (pre-planned deterministic) set of tactics, depending on the vessel's (perceived) information about the enemy position and intentions.

TASMAN is documented in Reference (8). An unclassified version of Reference (8), which omits the classified TASMAN data files, was prepared as Reference (9). The program was distributed to all TTCP nations. Canada (DMOR) converted the program from VAX FORTRAN to HP Multics FORTRAN. In doing so, Canada found and fixed several bugs in the program.

Later Michele Fisher of DMOR modified TASMAN so that it included the use of active sonar by the escorts, by the submarine and by the sonobuoys dropped by the helicopters. This modified version is described in Reference (10). The new version of TASMAN also included a 'machine independent' random number generator, due to Marse and Roberts, described in References (11) and (12).

Once the DND Multics system disappeared, it was necessary to transfer the TASMAN program back from the Multics to the DMOR MicroVAX. This was done by 'a contractor' (presumably Frank Fitch). Reference (13) briefly describes how to run the two versions of TASMAN ('base case' with passive sonar, and 'active case' that included active sonar).

TASMAN FILES

Reference (8) gives a very detailed account of all FORTRAN and data files for the 'base case'. No source code is included, but the values of all input data are given.

Reference (10) gives an equally detailed account of the additional FORTRAN and data files that are needed to run the 'active' version of TASMAN. No source code is given, nor are any input data values.

Reference (13) explains how these files are used to run the 'base' or 'active' version of TASMAN. Since it takes a long time to run TASMAN for a reasonable number of iterations (say 100), batch runs are done, using .com files, (test2.com for a 'base' run and test3.com for an 'active' run).

VAX directory [.tasman] contains files test2.com and test3.com. It also includes the subset of input data files that are to be used for the current run.

Directory [.tasman.fort] includes the Fortran and object modules for all routines (for both TASMAN versions), and the two .exe files. It also includes some .opt files that specify the object modules to be linked for each version.

Directory [.tasman.data] includes the input data files for all variations of both TASMAN versions.

CONTENT OF THE BATCH FILES

test2.com starts by moving to the 'working directory' [.tasman]. It then copies the appropriate subset of the input data files in [.tasman.data] into 'working' input files in [.tasman]. For instance, if we wanted to do 100 iterations we would copy [.tasman.data]dinitle.dat into [.tasman]dinit.dat.

Next, test2.com moves to directory [.tasman.fort] and links the appropriate object modules (those contained in file test2.opt). (Presumably this is done in case any of the FORTRAN routines have been recompiled since the previous run).

After returning to the working directory [.tasman], test2.com runs program [.fort]test2. This produces output in files rtam.dat and rtam1.dat. These two files are renamed rtam002.dat and tam002.dat.

Finally test2.com runs programs [.fort]proctam and [.fort]sumtam, which use the TASMAN output files to create an event list in file TDBnnn.dat and a summary table of results in file TSPnnn.dat. (Here 'nnn' denotes the run number, e.g. 003).

test3.com works in the same way as test2.com, but uses a different subset of input data files, and links a different set of object modules.

RUNNING THE PROGRAM

There was no difficulty in running either version of TASMAN as a batch job, by using

submit test2 or submit test3

A trial run of test2, using one iteration, took 59.21 sec. Thus a 100 iteration run would take about 100 minutes, and a 400 iteration run would take about 400 minutes, or 6 hours 40 minutes.

We had a look at the program, to see whether there was any easy way to make the program run faster. It turned out that the invocation of the random number generator could be improved, as explained in the next section.

THE TASMAN RANDOM NUMBER GENERATOR

As mentioned earlier, the DMOR versions of TASMAN use a 'machine independent' random number generator, which is discussed in References (11) and (12). Reference (12) states that in a 400 iteration run of the base case of TASMAN, the random number generator is called 37,440,000 times! In other words there are 93,600 calls per iteration.

The TASMAN random number generator 'rndmno' produces a value uniformly distributed between 0 and 1. However, most of the time TASMAN is looking for a random normal deviate. Examination of the program shows that random normal deviates are generated by a routine 'getsam' - and the deviates are produced by the (somewhat archaic) process of adding 12 'uniform' random numbers. (This method was used in the IBM Scientific Subroutine Package in the early 60's).

A test showed that the TASMAN routine 'rndmno' was somewhat slow. On the DMOR MicroVAX it took 19.16 seconds to generate 100,000 uniform random numbers, using 'rndmno'. Thus the 93,600 random number calls in a one-iteration run would take 17.93 seconds - out of 59.21 seconds for the whole run. In other words, the random number calls took just over 30% of the total run time.

It would seem a lot faster to generate a random normal deviate by generating one uniform random number variable, rather than twelve, and converting it to a random normal deviate by using an approximation to the inverse of the normal probability integral. Moreover, the sum of 12 independent uniformly distributed variables is not a very good approximation to the normal distribution. (See Reference (14)).

Mr. Emond of DMS furnished a routine 'ppnd7' (shown in Listing 1) which converts a random uniform variable into a random normal deviate.

We added 'ppnd7' to the TASMAN fortran file 'frand.for'. We also changed routine 'getsam' (found in TASMAN file 'ffun.for), by replacing the chunk of code from statement 11 to statement 10 by

```
11    getsam = rndmno(iseed)
      getsam = ppnd7(getsam,ixx)
```

where ixx was declared as INTEGER.

After recompiling and re-linking the test2 modules, we ran program test2 for one iteration. It ran in 44.88 seconds - some 24% faster than the 'old' test2 program.

We therefore made similar changes to routine 'getsam' in file 'ffunaa.for', and recompiled and linked the modules used in the 'active' program test3.

RUNNING THE NEW VERSIONS

We now made batch runs of the revised test2 and test3, both for 25 and for 100 iterations. Results were as follows.

Program	No. of iterations	Run time
test2	25	28m 7.95s
test3	25	45m 19.61s
test2	100	92m 49.70s
test3	100	185m 46.52s

It appears that a run of test3 takes about twice as long as a test2 run. This is because test3 is run with a time-step of 2 minutes, whereas the time-step for test2 is 5 minutes.

As a matter of interest, we also did a 25-iteration run of test3.com using the 'old' version of the random normal deviate routine getsam. The run took 65m 1.08s, compared with 45m 19.61s for the 'new' getsam version. Thus by using the new routine, we save just over 30% of the running time.

ERROR IN TEST2 PROGRAM

In carrying out one of the runs of test2, we found that the program aborted with an error message 'floating point divide by zero'. This error was traced to the routine 'rtshp' in file fhtac2.for, at the statement

```
      if(timhc(ihc,ihcst(ihc)).lt.dst/sphc(ihcst(ihc))+5.0)
     & retshp=.true.
```

We replaced this statement by the safer (equivalent) statement

```
      if(dst.gt.(sphc(ihcst)*(timhc(ihc,ihcst(ihc))-5.0)))
     & retshp=.true.
```

LIMITATIONS OF TASMAN

The present version of TASMAN has certain limitations.

(a) The number of escorts is limited to three or less. It would be possible to modify the program so that, say, four escorts could be used, but this would involve a fair amount of work. (See Appendix A).

(b) The tactics of the submarine, the convoy, the escorts and the helicopters are rigidly defined. Extensive modifications to the program would be needed to allow the user to choose from a variety of tactical doctrines.

(c) An open ocean environment is assumed.

(d) A single scenario is assessed.

ADVANTAGES OF TASMAN

The main TASMAN program FTAM, with its associated routines, produces 'raw' output in the form of a chronological event list stored in file RTAM1.DAT. However, post-processing program PROCTAM and SUMTAM can be run to use the event list to produce summary tables showing the results of the simulation, in file TSPnnn.DAT. (Here 'nnn' is the run number, for example '003').

In addition a pre-processing batch file (test2.com or test3.com) is used to select the subset of input files required for the current simulation, to link the appropriate routines, and to run programs FTAM, PROCTAM and SUMTAM.

CONCLUSION

This short Note deals with some runs of program TASMAN (both passive and active versions) on the DMOR MicroVAX, and discusses the effect of a slight modification to the TASMAN routine that generates random normal deviates. It appears that this modification results in a substantial reduction in running time.

Clear detailed explanations of the two versions of TASMAN are given in References (8) and (10). Moreover the FORTRAN source routines are well provided with explanatory comments.

The program is so detailed that it takes a long time to run any worthwhile number of iterations. It would run faster if it were transfered to the Alpha VAX, to the SUN or to the IBM 486.

Since TASMAN is at present being superseded by ANSWER, it is not clear how much effort it would be worth putting into future improvements in TASMAN.

APPENDIX A

MODIFICATIONS NEEDED TO ENABLE TASMAN TO BE RUN WITH FOUR ESCORTS

CHANGES NEEDED

In TASMAN the number of escorts NTA is read in from input file DINIT.DAT. In the standard 'base case' the number of escorts is read in as 3.

If we wish to use TASMAN to assess a case in which 4 escorts are used, it is not enough merely to change the value of NTA in file DINIT.DAT from 3 to 4. We need to make further changes of three types:

(a) Changes to numbers of various types of object in the simulation.

(b) Changes to arrays read in from the input data files.

(c) Changes to the dimensionality of arrays used in the simulation.

CHANGES TO NUMBER OF OBJECTS

Since each escort carries one helicopter, we must change the number of helicopters (NHCS, read in from file DINIT.DAT) from 3 to 4.

Once NTA and NHCS have been read in, TASMAN routine FTAM calculates values for

NSS (the number of surface ships),
NPF (the number of platforms - escorts, convoy and submarine),
NSP (the number of platforms with sonars),
NTA2CV (the number of platform detection opportunities),
NSN (the number of sonar sets on submarine and escorts)

as follows:

NSS = NTA + NCV NPF = NSS + 1 NSP = NTA + 1

NTA2CV = NTA + NSS NSN = NTA*NTASN + NSBSN

Here NCV = 1 is the 'number of separate ships' in the convoy, while NTASN and NSBSN (read in from DINIT.DAT) are the number of sonar types on an escort and on the submarine.

Naturally, as soon as you change the value of NTA, the values of NSS, NPF, NSP, NTA2CV and NSN all change.

CHANGES TO INPUT ARRAYS

The TASMAN input data files contain a large number of 'array variables' whose values have to be specified

(a) for each escort from 1 to NTA, or

(b) for each surface ship from 1 to NSS, or

(c) for each 'platform detection opportunitiy' from 1 to NTA2CV, or

(d) for each sonar set from 1 to NSN, or

(e) for each helicopter from 1 to NHCS.

Since we have increased the value of NTA from 3 to 4, with consequent changes to NSS, NTA2CV, NSN and NHCS, we have to specify values for the 'new' element or elements of each of the input arrays in (a), (b), (c), (d) and (e) above, and insert these new values in the input data files. (In the TASMAN 'base case', about 94 arrays would have to be modified in this way).

CHANGES TO DIMENSIONS OF ARRAYS

In TASMAN, arrays are dimensioned by using variables (such as II, III, IIII, JJJ and JJJJ) whose values are set in FORTRAN PARAMETER statements. For example, all arrays with NTA elements are dimensioned to size II, and II is set to 3 in a parameter statement.

But this is not the whole story. Instead of setting II = 3 in one parameter statement, TASMAN chooses to have a separate parameter statement in each of the 20 routines in which II is needed. In other words, if you want to change the dimensionality of all arrays of NTA elements from 3 to 4, you have to change 'II=3' to II=4' in each of 20 parameter statements!

The situation is even worse with some of the other dimensionality parameters. We summarise it below.

Parameter variable	Present value	Modified value	Array variable	No. of routines affected
II	3	4	NTA	20
III	4	5	NSS	63
IIII	7	9	NTA2CV	40
JJJ	5	6	NSN	8
JJJJ	3	4	NHCS	38

The above Table shows the very large number of parameter values that would have to be modified to effect the change from 'three escorts' to 'four escorts'.

Of course, the job would have been a lot easier if TASMAN had chosen to
set up an INCLUDE file, say PARAM.INC, that contained just a single PARAMETER statement that set the values of all the parameter variables, and include this INCLUDE file in each TASMAN routine that needed the value of one or more of the parameter variables.

CONCLUSION

This Appendix has discussed the changes that would have to be made to the TASMAN FORTRAN routines and input data files to allow one to run a simulation with 4 escorts instead of 3. One can see that this change would involve a fair amount of work.

LISTING 1

```
      real function ppnd7 (p,ifault)
c
c        ALGORITHM AS241    APPL. STATIST. (1988) VOL. 37, NO. 3
c
c        Produces the Normal deviate z corresponding to a given lower
c        tail area of p; z is accurate to about 1 part in 10**7.
c
c
      real zero, one, half, split1, split2, const1, const2,
     &  a0, a1, a2, a3, b1, b2, b3, c0, c1, c2, c3, d1, d2,
     &  e0, e1, e2, e3, f1, f2, p, q, r
      parameter (zero = 0.0e0, one = 1.0e0, half = one/2.0e0,
     &  split1 = 0.425e0,    split2 = 5.0e0,
     &  const1 = 0.180625e0, const2 = 1.6e0)
c
c     coefficients for p close to 1/2
      parameter (a0 = 3.38713 27179e0,
     &           a1 = 5.04342 71938e1,
     &           a2 = 1.59291 13202e2,
     &           a3 = 5.91093 74720e1,
     &           b1 = 1.78951 69469e1,
     &           b2 = 7.87577 57664e1,
     &           b3 = 6.71875 63600e1)
c
c      coefficients for p neither close to 1/2 nor 0 or 1
      parameter (c0 = 1.42343 72777e0,
     &           c1 = 2.75681 53900e0,
     &           c2 = 1.30672 84816e0,
     &           c3 = 1.70238 21103e-1,
     &           d1 = 7.37001 64250e-1,
     &           d2 = 1.20211 32975e-1)
c
c      coefficients fo p near 0 or 1
      parameter (e0 = 6.65790 51150e0,
     &           e1 = 3.08122 63860e0,
     &           e2 = 4.28682 94337e-1,
     &           e3 = 1.73372 03997e-2,
     &           f1 = 2.41978 94225e-1,
     &           f2 = 1.22582 02635e-2)
c
```

LISTING 1 (CONT.)

```
      ifault = 0
      q = p - half
      if (abs(q) .le. split1) then
         r = const1 - q * q
         ppnd7 = q * (((a3 * r + a2) * r + a1) * r + a0) /
     &                ((b3 * r + b2) * r + b1) * r + one)
         return
      else
        if ( q .lt. 0) then
           r = p
        else
           r = one - p
        endif
        if (r .le. zero) then
           ifault = 1
           ppnd7 = zero
           return
        endif
        r = sqrt(-log(r))
        if (r .le. split2) then
           r = r - const2
           ppnd7 = (((c3 * r + c2) * r + c1) * r + c0) /
     &              ((d2 * r + d1) * r + one)
        else
           r = r - split2
           ppnd7 = (((e3 * r + e2) * r + e1) * r + e0) /
     &              ((f2 * r + f1) * r + one)
        endif
        if (q .lt. 0) ppnd7 = -ppnd7
        return
      endif
      end
```

ANNEX F

PARTIAL REVIEW OF PROGRAM MCS

INTRODUCTION

MCS is an event-driven simulation of the clearance of a minefield at sea by a number of minesweeping vehicles. The operation is carried out by ships called Command Platform (Search), that search for and attempt to identify and locate individual mines, by Remote Search Vehicles and by Disposal Vehicles that detonate the mines. There can be 'non-mines' (mine-like objects) in the field. The simulation can include a Route Survey phase in which the field is surveyed before any mines are laid, and 'non-mines' are located.

The program is written in FORTRAN for the VAX. The main controlling program is CL100.FOR, and there are a large number of subroutines. The simulation includes events of 77 different types. As the simulation proceeds events are generated, placed in chronological order on a linked list, processed one by one and deleted.

The simulation can be run for a specified number of replications, or until a specified percentage of the mines have been destroyed.

A separate program CS100.FOR is used to generate data on the performance of the sonars to be used in MCS, under specified environmental conditions. In MCS, information on the characteristics of the minefield and of the clearance vessels is obtained by asking the user a series of 308 questions.

The output from MCS is stored in a file (run name).SOF. This file can be printed to obtain a tabular record of the results of the simulation. Post-processing programs MCSPLT.FOR and MCSGRPH.FOR can be used to plot minefield data and vehicle movements, if appropriate plotting devices are available.

MCS was developed by British Aerospace in the 1980's, and was used by Fleet Technology, under contract to DND, to run several simulations pertinent to Canadian mine clearance situations. Runs were also made by DMOR and by EDRD (formerly DREP). (See References (15), (16) and (17)).

REVIEW OF MCS

In mid-June 95 an attempt was made to locate the MCS source programs. Searches were made in the DMOR and EWORT Shields. One VAX tape labelled 'FLEET-HUNT' was found, but it proved to be empty. Dr. Preston of EDRD was asked whether he had MCS source programs. He said that he had source for MCS Version 4, but that this version was hard to use, and that it was difficult to obtain reliable output. He also sent DMOR details of many bugs found in Version 4, most of which had been corrected in later versions. He recommended that DMOR obtain MCS Version 11 from British Aerospace, since this version was (apparently) free of bugs and it was much easier to use. He was reluctant to supply MCS Version 4 to DMOR.

Dr. Glen, President of Fleet Technology said that Fleet no longer had any MCS source. After further search, another two VAX tapes labelled 'FLEET' and 'HUNT' were finally found. After considerable difficulty it was found possible to read these tapes and transfer their contents to the DMOR VAX. The HUNT tape was transferred to VAX directory [HUNT] and various sub-directories. There were 475 source and data files, some of which were duplicates.

The whole process of searching for the source files, finally locating them, and reading them on to the DMOR VAX, took six weeks!

KERMIT was used to transfer the source files to the IBM PC, where they were printed out. No attempt was made to compile and run MCS on the PC, because the VAX version made use of several commands specific to the VAX terminal.

On the VAX, the MCS source files were examined, and it seemed that all the bugs reported by Dr. Preston had been corrected. The files were succesfully compiled and linked. It proved to be impossible to run MCS using the input data files found on the HUNT tape, since none of these sets of data were mutually compatible, and attempts to run resulted in aborts.

Finally the sonar program CS100 was used to generate input files based on a hypothetical sonar system. MCS was then run, using a run name DMOREL, and sonar data files called DMORE1. This time the program accepted the input data as compatible, and began to run. However it aborted with a message 'Processing Event 13 - EVENT NUMBER OUT OF RANGE'.

A large number of additional WRITE statements were inserted into program module SC205 that controlled the progress of the simulation, and into routines UD206 and LOCATE of module MCSLIB that processed the linked list of events, also into routine EVNT13 of module SE206. The simulation was run again. It still aborted while processing event 13.

Further runs were made. All aborted, some at event 13, others at events 39 and 48. (All these were events that concerned classification of mine detections). Moreover the output files DMOREL.SOF, which listed the mine positions, the movements of the ships and the events that occurred, did not seem to show a logical pattern.

Incidentally a minor improvement was made to the MCS routine NORMAL, that generated random normal deviates by taking the sum of six 'independent' uniform random numbers. This 'sum of six' variable is a very poor approximation to a normal deviate. (See Reference (18)). Instead NORMAL was modified so as to generate one uniformly distributed variable, and routine PPND7 was used to convert this to a random normal deviate. (See Listing 1 of Annex E for a listing of PPND7). This process is a lot faster and more accurate than the old procedure.

In view of the considerable time spent on the above MCS work, and of the limited time remaining in this Contract, at DMOR's direction further work on MCS was abandoned. (See Listing 1 of Annex E for a listing of PPND7).

CONCLUSION

The above Annex has described the work done on MCS between mid-June 1995 and mid-Sept 1995. Since it was not possible in the time available to produce a running version of MCS, at DMOR's direction further work on MCS was abandoned.

ANNEX G

REVIEW OF PROGRAM BPM

INTRODUCTION

The Barrier Patrol Model (BPM) was developed by DMOR staff between 1982 and 1992. The Model simulates an encounter between a 'Blue' submarine patrolling a barrier line, and a 'Red' submarine that attempts to penetrate the barrier. The Blue submarine patrols his barrier either

(a) in a straight line 'back-and-forth' pattern, or

(b) in a 'snakelike track' to and fro, or

(c) in a 'racetrack' pattern.

'Red' tries to penetrate the barrier either

(a) in a straight line track at a random course angle that intersects the barrier, or

(b) in a 'snakelike' approach pattern.

In the early versions of BPM it was assumed that both sides could use passive towed-array sonars and hull mounted sonars, at various frequencies.

The game ends when the Red submarine penetrates the barrier, or when either submarine detects his opponent within some critical range at which, it is assumed, the aggressor can close to destroy his target. (Later versions of BPM that include target classification and Target Motion Analysis have been proposed, but never produced).

The early versions of BPM that used only passive sonars, were developed by Mr. Bender and Miss Provencher of DMOR. They were written in FORTRAN for the HP Multics system then in use by DND.

Later Dr. H. Yip of DMOR modified BPM so that both sides could use active sonars. He also changed the code so that the program could be run on an IBM PC rather than the Multics computer, which by then had disappeared from DND. This entailed several changes including the replacement of the Multics 36-bit random number generator by a 32-bit routine. Dr. Yip chose to use the 'RAN2' routine from 'Numerical Recipes' (see Reference (6)). Since BPM made use of enormous amounts of memory, it was necessary to compile and link the program using Lahey FORTRAN, that has the 'DOS Extender' feature which handles programs that require more than 640K of memory. (BPM uses about 1.45 megs!).

REVIEW OF BPM

As part of its current DMOR Contract, Domus Software was asked to review BPM. In mid-June 95 Dr. Mitchell produced six floppy disks that contained routines and data files relating to BPM. Three of the disks contained the BPM version written by Miss Provencher. The other three dealt with Dr. Yip's revised version.

These last three disks were examined. The most useful file on the first disk (which we will call 'DISK 0'!) is a small READ file that explains what is on the other two disks, called 'DISK 1' and 'DISK 2'. DISK 1 contains the following directories:

(a) \envn_1-5 - acoustic data for 5 different environments.

(b) \vangle - vertical arrival angle data.

(c) \code - all source code and data files needed to run BPM.

DISK 2 contains the following:

(a) \hla - signal excess data for HLA.

(b) \vla - signal excess data for VLA.

(c) \absin - all batch ('absentee') files for running BPM.

DISK 1 also contains a very useful batch file BPINST. This creates all the necessary sub-directories on 'drive C', and then copies the complete contents of DISK 1 into the appropriate sub-directories of drive C. For example, if DISK 1 is in drive A and the 'target drive' on which you wish to run BPM is drive C, you invoke BPINST by typing

BPINST A C

Next, you insert DISK 2 into drive A and use batch file BPINST2 to tranfer the contents of DISK 2 into the appropriate sub-directories of drive C by typing

BPINST2 A C

BPM can now be compiled, linked and run by using batch file RUN.BAT, which contains the following:

```
f7713 wkbarrie.for
f7713 wknoise.for
386link wkbarrie.obj wknoise.obj -exe wkbarrie.exe
wkbarrie <ela1.abs >ela1.out
```

The first two lines of RUN use Lahey FORTRAN to compile the two BPM source files 'wkbarrie' and 'wknoise'. ('wknoise' contains routines that calculate the noise produced by wind and distant shipping). The next line uses Lahey FORTRAN to link the programs. (Lahey automatically assigns enough memory to run 'wkbarrie').

The last line runs BPM, using input data in batch file ela1.abs and storing those output statements that usually appear on the terminal in file ela1.out. The actual BPM output is stored in the output file named in ela1.abs - in our case, barnel.out. This file contains a repeat of the input data, followed by tables that show the number of contacts achieved within each 10 nm cell of range, and distributions of the durations of each contact held. (Some auxiliary routines enable one to produce histograms of the output data, if required).

A run of BPM for a typical case, for 100 replications, took about 9 minutes.

COMMENTS

Of course, the first thing to be said about Dr. Yip's version of BPM is that it runs correctly, in a reasonable time. However, one or two minor comments may be in order.

(A) To an old-fashioned programmer like the writer, the lavish and wasteful use of computer memory by this program makes one cringe. The executable file occupies about 1.45 megs, of which about 1.25 megs is needed to store the COMMON arrays.

For example, each of the arrays histbl() and histrd() has 49,600 elements. The arrays are declared as INTEGER (i.e. INTEGER*4), so that these two arrays alone occupy almost 400 kb of memory. But histbl() merely indicates the detection status of Blue against Red for various sonars at various frequencies for each time step of the simulation run. It can only have values of 'zero' or 'one', and thus could easily be declared as BYTE or INTEGER*1. This would save about 300 kb of memory.

A more ruthless treatment would declare all the remaining arrays as INTEGER*2, thus quantising their values to one of 32,768 amounts. This would save about another 400 kb of memory, but would slightly complicate the source code.

Presumably the modern programmer would say - 'There is plenty of memory available, so why bother?'. In which case, the above suggestions are irrelevant.

(B) Random normal deviates are generated by the hoary old process of adding 12 'independent' uniformly distributed random numbers and using the sum as a random normal deviate. As shown in Reference (14), this method is not very accurate.

It would seem better to use the same method as that suggested for TASMAN - generate one uniform random value and convert it into a random normal deviate by using routine PPND7. (See Annex E, Listing 1). This process is much faster and more accurate than the 'sum of 12 uniforms' method.

(C) The printout of results generated by routine OUTSTAT1 uses lines of 132 character length - obviously a hangover from Multics. This produces a confusing and hard-to-read output on an '80-character per line' printer. The output statements in OUTSTAT1 should be revised to conform with '80 characters per line'.

(D) BPM is a 'time-step' simulation. It would probably run faster if it were set up as an 'event-driven' simulation. However, this would require extensive re-writing of a lot of code, and it would perhaps be quite difficult to adapt the 'Seymour' detection model to an 'event-driven' procedure.

(E) Ample comments are provided within the program, and the COMMON variables are fairly well defined. However there seems to be no external documentation that describes the 'theory' of the program. (Such a document is referred to in one of the README files on the floppy disks, but no such document has been located).

BPM has been successfuly modified to incorporate (B) and (C). But since BPM now runs successfully in an acceptable time, there seems to be little point in further modification - especially since DMOR has available the ANSWER simulation which presumably could handle the BPM problem.

The revised program is stored on diskette G, labelled BPM.

ANNEX H

NOTES ON SACLANT THREAT MODEL

INTRODUCTION

The SACLANT Threat Model provides an assessment of the total stockpile of ordnance required, in ships and at resupply sites, at the start of an operation to ensure that a (specified) 'adequate proportion' of all the attacking targets will be destroyed (or to ensure that some other Measure of Effectiveness will attain a specified value). The main assumptions made by the model are the following:

(A) For a given weapon type and a given target type, a certain specified number of weapons is needed to destroy all the targets of that type.

(B) All ships have the same initial allowance (IA) of ordnance.

(C) There is a single resupply facility that supplies all ships.

(D) When a ship has expended a certain number of weapons (R), the ship requests another R weapons from the resupply site.

(E) No ship resupplies another ship.

(F) There is a specified total number of targets. These targets are distributed among the ships according to a Polya distribution with given parameter ß. (See Appendix A for a description of the Polya distribution).

(G) Regarding resupply, if at the end of the operation the ships have made more requests for 'supply chunks of R rounds' than can be handled by the depot, the available 'chunks' are assumed to have been distributed randomly among the ships. (Note that it is <u>not</u> assumed that ships are resupplied in the order in which they make requests. In fact, 'time' is never considered in this model - only the state of the system at the end of the operation).

(H) If because of supply shortage the total number of rounds fired is less than the specified number of rounds needed to destroy all the targets, the rounds are assumed to be assigned randomly among the targets. This leads to a hypergeometric distribution for the total number of targets destroyed.

The program determines two Measures of Effectiveness:

(a) The average proportion of targets destroyed.

(b) The average proportion of ships that do not run out of ordnance by the end of the operation.

The model was developed for the U.S. Navy, under contract by QUANTICS. Reference (19) gives a detailed account of the model, with proofs of the various algorithms used in deriving the Measures of Effectiveness.

GENERAL DESCRIPTION OF MODEL

Reference (19) first describes the Polya distribution, and shows that this distribution is a good fit to the distribution of targets killed among combatants in a number of practical cases, such as Air-to-Air Kills in Vietnam, Submarine Kills of Ships in World War II in the Pacific and Atlantic Theatres, and Tank Kills of Tanks in the 1973 Arab-Israeli War. The values of the Polya parameter ß that fit these practical data vary from 0.45 to 1.10. (Reference (19) notes that if one can assume that ß = 1, the mathematics becomes very much simpler!).

Next Reference (19) assumes that the total number of rounds (say d) required to kill all the targets is distributed among the ships in a Homogeneous Polya distribution with parameter ß. If a ship expends R rounds, rounds are resupplied from the supply site in 'chunks' of size R. If there are not enough rounds to satisfy the demands of all the ships, 'resupply chunks' are allocated randomly to ships.

Some ships may not have enough rounds to engage all the targets for which they are responsible. Thus the total number of rounds fired by all ships, say s, may be less than d. Reference (19) develops a formula that gives the probability distribution of s. (In Reference (19) the quantity d is referred to as the total amount of ordnance 'demanded' by the targets, while s is called the total number of rounds 'supplied up' to the targets).

Reference (19) also derives a formula for the probability distribution of r, the number of ships that have run out of ordnance by the end of the operation.

It is assumed that the s rounds available are distributed at random among the T targets. On this assumption Reference (19) deduces a formula for the probability distribution of t, the number of targets killed.

Formulae are also derived for the expected values (and variances) of s, r and t. Appendix B gives a very brief explanation of the derivation of the above formulae.

Note that there is no direct allowance for attrition of ships during the operation. This matter is discussed in Reference (19), which asserts that for the attrition model considered there, the effect of attrition can be 'built into' the distribution of targets killed among ships, so that this distribution is still of 'Polya' form.

The (perhaps somewhat dubious) reasoning behind this assertion is contained in Appendix 2D of Reference (19). This matter is rather briefly discussed in Appendix C.

MONTE CARLO METHOD

Having gone through a considerable amount of combinatorial mathematics to develop all these formulae, Reference (19) points out that for practically useful values of T, d and n (the number of ships), the number of operations needed to compute the required Measures of Effectiveness is so great that the calculations are well beyond the capabilities of modern computers. As usual in such cases, the author resorts to a Monte Carlo method of getting an approximate solution to his problem.

The Monte Carlo process requires the values of the following quantities:

n - number of ships,

T - total number of targets,

d - total number of rounds needed to kill all T targets,

IA - initial number of rounds carried in each ship,

R - number of rounds in a 'resupply chunk',

m - total number of 'resupply chunks' in the supply site,

ß - the 'Polya parameter'.

Most of these quantities are program input data, but two (d and m) have to be calculated from other input data. A complete list of all the input data is given in Appendix D.

The Monte Carlo method first generates a sample assignment of 'rounds required' d among the individual ships, on the assumption that the distribution is 'Homogeneous Polya' with parameter ß. (This assigment is carried out by 'tossing the rounds into ships' as described in Appendix A).

For this sample distribution of d among ships, one calculates the number of 'replacement chunks' that each ship needs to engage all its targets. If the supply site cannot cope with all the replenishment requests, the site allocates its 'chunks' at random among the ships.

One can then calculate how many rounds each ship is able to fire, and hence deduce the values of s, the total number of rounds fired, and r, the number of ships that ran out of ordnance by the end of the operation.

Further, by assigning the s rounds randomly among the T targets, one finds out how many targets (t) are killed in this 'sample run'.

By repeating this process many times, one gets an approximate idea of the average values of s, r and t.

COMPUTER PROGRAM

The floppy disk supplied with the SACLANT Threat Model contains a series of files that make up a FORTRAN program which carries out the Monte Carlo calculations mentioned above. The files are discussed below. Figure 1 is a summary flow diagram that gives a general idea how the principal routines interact to produce the required results. The files are:

INPUTBRF. This contains the driver routine THREAT, the main calculating routine MODEL, and a series of input routines including CONSOLE, which gets input parameters from the computer terminal, and READ_IN, which could get inputs from a disk file. The input data include the number of targets of each of a number of 'target types', and the number of 'shooters' of each of a number of 'shooter types'. These data are given for each of a number of 'years'. (A 'year' constitutes one operation). Other data give the kill capability of each shooter type against each target type, and the initial load and reorder 'chunk size' for each type of shooter. As mentioned above, Appendix D gives a description of all the input data.

File INPUTBRF also contains routine ASSIGN, which assigns various shooter types to various targets. A few more routines then compute, for each shooter type for each 'year', the expected number of rounds required to kill all the targets assigned to this shooter type. (In other words, the quantity 'd'). Routine MODEL then calls a routine, quaintly called MAIN, which is in another disk file MONTEBF2.

MONTEBF2 contains routine MAIN, which controls the calculations of Measures of Effectiveness. After a few preliminary computations control is passed to routine MONTE, which handles the Monte Carlo assessments. First, routine BOX generates a sample distribution of d (rounds required to kill all targets) among the individual ships, according to a Polya distribution with parameter ß. Next REFILLS generates a random allocation of the 'resupply chunks of ordnace' to the ships, after which ANALYZE calculates the number of ships that run out of ordnance for this sample run.

This process is repeated for some specified number of iterations, after which PRNOUT calculates the expected proportion of ships that run out.

All these calculations are repeated for a number of values of the initial stock level at the resupply site (i.e. initial inventory).

Finally the program estimates the initial inventory (INV) needed at the resupply site to ensure that the Measure of Effectiveness (MOE, expected proportion of ships that do not run out) has some specified value. It does this by interpolation from the previously calculated series of values of MOE corresponding to various values of INV. The interpolation routines are held in file INTERP.

File INTERP contains a series of routines that carry out inverse cubic interpolation according to a method developed by Fritsch and Butland (Reference (20)).

After the interpolations are done, back in file MONTEBF2, routine SPLITREQ writes out the results, either to the terminal or to a predesignated file.

File ROG RAND contains a random number generator RANDM used in the Monte Carlo assessment.

To execute the program, type

MONTEBF2

and follow the instructions on the terminal screen.

TROUBLES IN COMPILATION

When we attempted to compile the component files of the SACLANT Threat Model, we found that the Microsoft FORTRAN compiler objected to the use of MAIN as the name of a subroutine, and a number of fatal error messages were produced.

The trouble was overcome by removing MAIN from file MONTEBF2, putting it in a separate file and renaming it MAINX. The modified file MONTEBF2 was renamed MONTEBF3. Compilation was then satisfactory, and the program could be run without trouble. Some typical runs, using 400 replications, each took only a few seconds on the IBM 486.

PREVIOUS WORK BY DMOR ON SACLANT MODEL

Dr. H. Yip has previously done work on the SACLANT Threat Model. He supplied a floppy disk that contained his work. His source programs were compiled and executed. His modification gave results that were substantially the same as those given by the original model.

LIMITATIONS OF PRESENT MODEL

First, the model only calculates one 'Measure of Effectiveness', the expected proportion of ships that have not run out of ordnance at the end of the operation. No attempt is made to compute the more pertinent Measure, the expected proportion of targets destroyed.

Second, although the model makes provision for calculating the inventories that correspond to six specified values of 'expected proportion of ships that do not run out', routine MOE is 'hard-wired' to compute only the inventory required to produce an MOE value of 0.99.

Memos previously sent by QUANTICS to DMOR state that model versions are available which compute the inventories required to produce specified values of either Measure of Effectiveness. (See Reference (21)). DMOR recently asked QUANTICS whether they could supply source files for these 'extended' model versions. (See Reference (22)). By mid-May QUANTICS had not replied.

A MONTE CARLO DIFFICULTY

There is a third difficulty with the model, that is caused by the use of the Monte Carlo method. Consider the case examined in Reference (19), page 15 (Exhibit 0-4). This shows the inventories required to ensure that the average percentage of targets killed is P, for various values of P, for the following case:

1000 targets; 50 ships; 2 rounds per target kill;
40 rounds initial allowance per ship;
resupply after 20 rounds fired.

Some of the results from the Reference (19) table are:

P (%)	96	98	99	99.9	99.99	100
Inventory	3113	3249	3343	3495	3571	3960

A Monte Carlo run of this case, using MONTEFZ with 5000 replications, gave results that were very close to those given above, except for P = 100%, wher the Monte Carlo run gave an inventory of 3760, instead of 3960!

Reference (19) explains that to achieve a P value of 100% the inventory must be sufficient to cover <u>all</u> distributions of targets among ships, including the extreme case in which one ship engages all 1000 targets. In this case the required inventory is 3960.

However, Appendix E to this Annex shows that an inventory of 3960 rounds is required not only when one ship engages all 1000 targets, but also when one ship engages 991 or more targets. This Appendix then calculates the chance that any one ship will engage 991 or more targets. This chance trurns out to be only

$$8.3723 \times 10^{-74}$$

It follows that a Monte Carlo run with 'only' 5000 replications is most unlikely to include the case in which one ship engages 991 or more targets, which is the only case in which an inventory of 3960 is needed. Thus such a Monte Carlo run is extremely likely to conclude that the inventory needed to have a 100% chance of destroying all the targets, will turn out to be less than 3960.

MODIFICATIONS TO MODEL

The program was slightly modified so that, instead of assessing the inventory required to ensure that the expected proportion of 'ships not running out' is 0.99, it will calculate the inventories that correspond to various specified expected proportions of 'ships not running out'.

Since by mid-May 1996 QUANTICS had not sent us a useful reply to our recent memo, the program was further modified so that it could use 'expected proportion of targets killed', instead of 'expected proportion of ships not running out', as the Measure of Effectiveness.

The resulting source files, now called MONTEBFZ.FOR, INPUTBRZ.FOR, MAINX.FOR, INTERP.FOR and ROG_RAND.FOR, as well as the executable program MONTEBFZ.EXE are stored on diskette I labelled 'SACLANT Threat Model'.

LATEST VERSIONS OF MODEL

At the end of May 1996 QUANTICS finally replied to the DMOR memo (Reference (23)). QUANTICS sent two diskettes, the first containing source and executable routines for versions MONTEFUL and SACMONTE of the SACLANT Threat Model. The second diskette contains a version of SACMONTE in which the inputs are provided by a Graphic User Interface (GUI), running under Microsoft Windows.

PROGRAM MONTEFUL

The more interesting of these two programs is MONTEFUL. This program calculates the total inventory needed to ensure that at least X% of the targets are killed (or that X% of the ships do not run out) with Y% confidence. Incidentally MONTEFUL also calculates the inventory needed to ensure that the average proportion of targets killed, or of ships not running out of ordnance, is some specified amount.

To calculate all these quantities by a Monte Carlo method, one needs to store intermediate results in some very large arrays. It turned out that the total amount of memory required to run MONTEFUL exceeded the 640K available for DOS, so that this program had to be compiled under Lahey FORTRAN using the DOS Extender.

MONTEFUL, as received, calculated these inventories for

X values of 70, 75, 80, 85, 95 and 99 percent, and

Y values of 50, 60, 70, 80, 90 and 100 percent.

The program was modified so that any desired set of X values could be used. The modified program, called MONTEFLX, is stored on diskette H.

PROGRAM SACMONTE

The other QUANTICS program, SACMONTE, is similar to the older program MONTEBF2, except that SACMONTE allows for the presence of a number of ships of a Combat Logistics Force (CLF), as well as the ships that fire ordnance against targets.

There is not sufficient time available to make a complete appraisal of SACMONTE, but cursory study of the source code suggests that the CLF ships merely receive ordnance from the single resupply source and then pass it on to the 'active' ships that engage targets. The SACMONTE output merely shows (a) the total inventory required to ensure that the average proportion of ships not running out of ordnance is 99%, and (b) the amount of such inventory supplied to each of the CLF ships.

The GUI version of SACMONTE is supposed to facilitate the input of data. An attempt was made to install and run this version of SACMONTE on an IBM PC running under Windows 3.1. The program was installed without difficulty, but when a program run was attempted, the result was a dismal failure. Either the GUI instructions were not clear, or there is some error in the code. Probably the former.

QUANTICS will be told about the GUI problems, and asked for clarification.

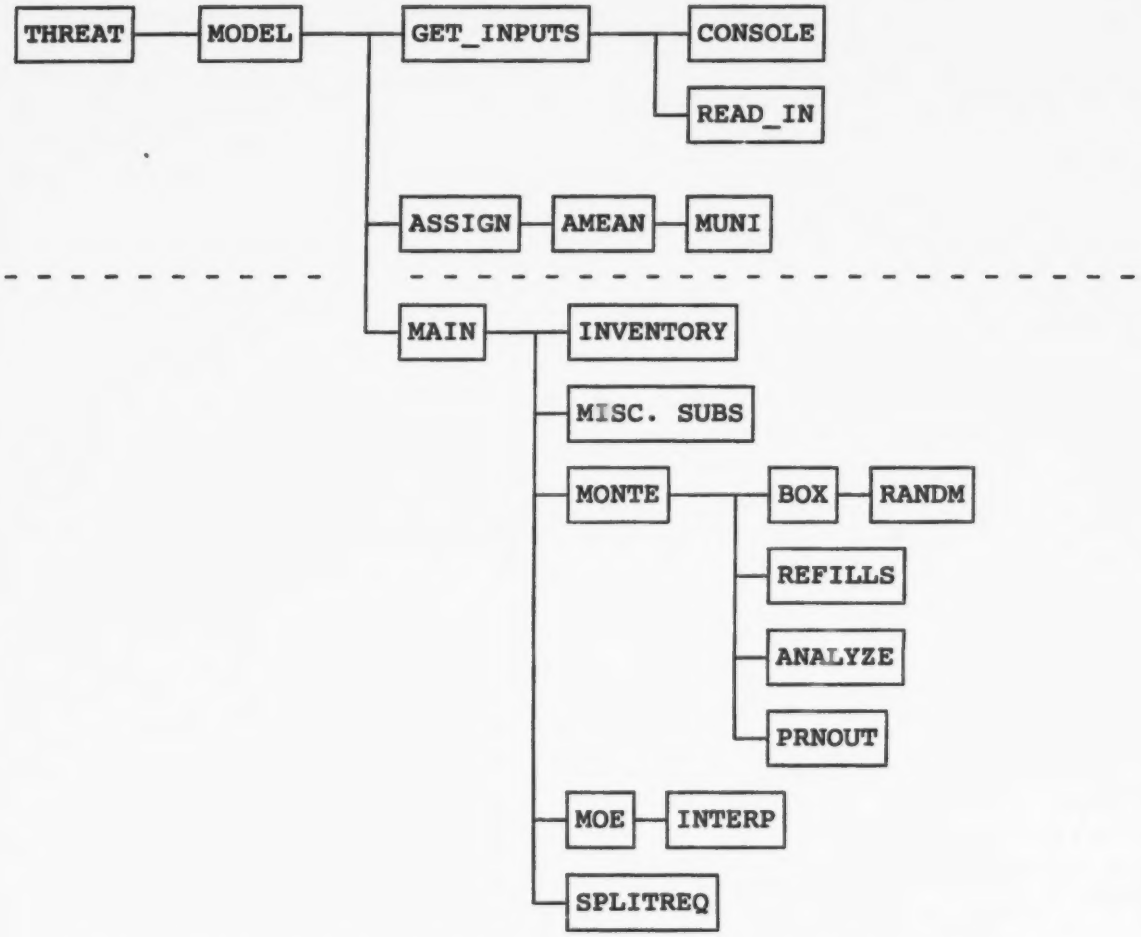
CONCLUSION

This Note has given a brief account of the SACLANT Threat Model, and of the associated computer programs. Modifications to the programs, to improve their usefulness, have been suggested and implemented.

FIGURE 1

SUMMARY FLOW DIAGRAM

FOR SACLANT THREAT MODEL PROGRAM



Note Routines above the dashed line are in file INPUTBRF. Those below the line are in file MONTEBF2, except for (a) routine INTERP, which is in file INTERP, and (b) routine RANDM, which is in file ROG RAND. Note that RANDM is called by several other routines in file MONTEBF2.

APPENDIX A

NOTE ON THE POLYA DISTRIBUTION

Suppose that we have n empty urns, of weights

$$\beta_1 , \beta_2 , \beta_3 , \ldots\ldots\ldots \beta_n$$

and total weight ω, where

$$\omega = \sum_{i=1}^{n} \beta_i$$

Let us toss a ball of unit weight, at random into one of the urns, so that the chance that the ball falls into a particular urn, say urn j, is

$$\frac{\text{current weight of urn j}}{\text{total of current weights of all urns}} = \frac{\beta_j}{\sum_{i=1}^{n} \beta_i}$$

The weights of the urns (with their contents) will now be

$$\beta_1', \beta_2', \beta_3', \ldots\ldots \beta_n'$$

where

$$\beta_j' = \beta_j + 1 \; ; \; \beta_i' = \beta_i \text{ for all } i \neq j$$

Now toss a second ball of unit weight into one of the urns, so that the chance that the ball falls into a specified urn, say urn k, is

$$\frac{\text{current weight of specified urn (with contents)}}{\text{Total of current weights of all urns (with contents)}} = \frac{\beta_k'}{\sum_{i=1}^{n} \beta_i'}$$

Continue tossing balls at random in this way, the chance that a ball falls into a specified urn being

$$\frac{\text{current weight of specified urn (with contents)}}{\text{current weight of all urns (with contents)}}$$

It can be shown that if d balls are tossed in this way, then the chance that urn 1 gets d_1 balls, urn 2 gets d_2 balls, urn 3 gets d_3 balls, ... and urn n gets d_n balls is

$$P(d_1, d_2, d_3, \ldots d_n) = \frac{\binom{-\beta_1}{d_1}\binom{-\beta_2}{d_2}\binom{-\beta_3}{d_3}\cdots\binom{-\beta_n}{d_n}}{\binom{-(\beta_1+\beta_2+\beta_3+\ \ldots\ \beta_n)}{d}} \qquad (1)$$

where

$$d_1+d_2+d_3+\ \ldots\ +d_n = d$$

and where, as usual,

$$\binom{n}{r} = \frac{n(n-1)n-2)\ \ldots(n-r+1)}{1.2.3.\ \ldots\ r}$$

and this relation holds whether n is positive or negative.

This is called the Polya distribution of d balls among n urns with parameters $(\beta_1, \beta_2, \beta_3\ \ldots\ \beta_n)$.

Equation (1) can also be written as

$$P(d_1, d_2, d_3\ \ldots\ d_n) = \frac{\binom{\beta_1+d_1-1}{d_1}\binom{\beta_2+d_2-1}{d_2}\binom{\beta_3+d_3-1}{d_3}\cdots\binom{\beta_n+d_n-1}{d_n}}{\binom{\beta_1+\beta_2+\beta_3+\ \ldots\ \beta_n+d-1}{d}}$$

In the special case in which all urns have the same initial weight ß we have

$$\beta_1 = \beta_2 = \beta_3 = \ldots = \beta_n = \beta$$

so that the Polya distribution becomes

$$P(d_1, d_2, d_3, \ldots d_n) = \frac{\binom{-\beta}{d_1}\binom{-\beta}{d_2}\binom{-\beta}{d_3}\cdots\binom{-\beta}{d_n}}{\binom{-n\beta}{d}}$$

This is called the Homogeneous Polya distribution. This is the distribution that is used in the SACLANT Threat Model to represent the distribution of d targets among n ships. The parameter ß is called the Polya parameter.

Note that in the case of the Homogeneous Polya Distribution the value of $P(d_1,d_2,d_3, \ldots d_n)$ does <u>not</u> depend on the order of the d's.

P499084.PDF [Page: 82 of 97]

APPENDIX B

DISTRIBUTIONS OF s, r AND t

Let there be n ships, each with initial armament load IR. When R rounds have been fired, the ship requests another R rounds from the resupply site, which initially holds mR rounds (m 'chunks' of R rounds each).

Let there be T targets, and let d be the total number of rounds needed to kill all the targets.

Let the d 'rounds needed' be distributed among the n ships in a Homogeneous Polya distribution with parameter ß, so that the chance of a certain distribution $\delta = (d_1, d_2, d_3, \ldots d_n)$ of the d rounds among the ships is

$$P_1(\delta) = \frac{\binom{-\beta}{d_1}\binom{-\beta}{d_2}\binom{-\beta}{d_3}\cdots\binom{-\beta}{d_n}}{\binom{-n\beta}{d}} \qquad (1)$$

Then the number of 'chunks of ordnance' of amount R requested by ship i will be

$$z_i(d_i) = \max\left(\left\lfloor \frac{d_i}{R} \right\rfloor, 0\right) \qquad (2)$$

so that the total number of chunks required by all ships is

$$z(\delta) = \sum_{i=1}^{n} z_i(d_i) \qquad (3)$$

DISTRIBUTION OF s

However, the resupply site only has m chunks available. If $z(\delta) \leq m$ the site can supply the needs of all ships. But if $z(\delta) > m$ the depot does not have enough chunks to resupply the needs of all ships. In this case it is assumed that the depot's available chunks are distributed <u>at random</u> among the ships (of course, without giving any ship more than it needs). It can be shown that on this assumption, for a given distribution $(d_1, d_2, d_3 \ldots d_n)$ of d among ships, the chance that ship 1 gets m_1 chunks, ship 2 gets m_2, ... is

$$P_2(m_1, m_2, m_3, \ldots m_n) = \frac{\binom{z_1(d_1)}{m_1}\binom{z_2(d_2)}{m_2}\binom{z_3(d_3)}{m_3}\cdots\binom{z_n(d_n)}{m_n}}{\binom{z(\delta)}{m}} \quad (4)$$

or, if we define the vector $(m_1, m_2, m_3, \ldots m_n)$ as Δ,

$$P_2(\Delta|\delta) = \frac{\binom{z_1(d_1)}{m_1}\binom{z_2(d_2)}{m_2}\binom{z_3(d_3)}{m_3}\cdots\binom{z_n(d_n)}{m_n}}{\binom{z(\delta)}{m}} \quad (5)$$

If ship i is supposed to fire d_i rounds against its targets, and if m_i chunks are resupplied to this ship, the number of rounds that the ship can fire is

$$s_i(d_i, m_i) = d_i \quad \text{if } z(\delta) \leq m \quad (6A)$$

and

$$s_i(d_i, m_i) = \min(d_i, IA + Rm_i) \quad \text{if } z(\delta) > m \quad (6B)$$

Hence for given δ and Δ, the total number of rounds fired by all ships is

$$s(\delta, \Delta) = \sum_{i=1}^{n} s_i(d_i, m_i) \quad (7)$$

P499084.PDF [Page: 84 of 97]

Thus if $z(\delta) \leq m$ we have

$$s(\delta,\Delta) = d \tag{8A}$$

while if $z(\delta) > m$

$$s(\delta,\Delta) = \sum_{i=1}^{n} \min(d_i, IA+Rm_i) \tag{8B}$$

Hence the chance that all ships fire a total of, say, s' rounds at the targets is

$$P_3(s') = \sum_{(\delta,\Delta)\in\Omega_3} P_1(\delta)\,P_2(\Delta|\delta) \tag{9}$$

where Ω_3 is the set of all (δ,Δ) for which

$$s(\delta,\Delta) = s' \tag{10}$$

DISTRIBUTION OF r

Similarly the number of ships that run out, given (δ,Δ), is

$$r(\delta,\Delta) = 0 \quad if\ z(\delta)\leq m \tag{11A}$$

or

$$r(\delta,\Delta) = \sum_{i=1}^{n} I(d_i, IA+Rm_i) \quad if\ z(\delta)>m \tag{11B}$$

where

$$I(x,a) = 0 \quad if\ x < a\ ;\ = 1\ otherwise \tag{12}$$

Hence the overall chance that, say, r' ships run out is

$$P_4(r') = \sum_{(\delta,\Delta)\in\Omega_4} P_1(\delta)\,P_2(\Delta|\delta) \tag{13}$$

where Ω_4 is the set of all (δ,Δ) for which

$$r(\delta,\Delta) = r' \tag{14}$$

DISTRIBUTION OF t

If d rounds are needed to kill all T targets, but only s rounds are fired, and if it is assumed that the s rounds are distributed randomly among the T targets, then the chance that t targets are killed is

$$\frac{\binom{s}{t}\binom{d-s}{T-t}}{\binom{d}{T}} \quad for\ t = 0,1,2,\ldots T \tag{15}$$

Thus the overall chance that t targets are killed is

$$P_5(t) = \sum_{s=0}^{d} \frac{\binom{s}{t}\binom{d-s}{T-t}}{\binom{d}{T}} P_3(s) \tag{16}$$

EXPECTED VALUES OF s, r AND t

From equations (9), (13) and (16) we see that

$$E(s) = \sum_{s'=0}^{d} s' P_3(s') \tag{17}$$

$$E(r) = \sum_{r'=0}^{n} r' P_4(r') \tag{18}$$

and

$$E(t) = \sum_{t=0}^{T} t P_5(t) \tag{19}$$

We can simplify equation (19). From (15) we see that the chance that t targets are killed, given s, is

$$P(t|s) = \frac{\binom{s}{t}\binom{d-s}{T-t}}{\binom{d}{T}} \tag{20}$$

Hence the expected value of t, for given s, is

$$E(t|s) = \sum_{t=0}^{T} tP(t|s) \qquad (21)$$

Thus from (16) the overall expected value of t is

$$E(t) = \sum_{t=0}^{T} tP_3(t) = \sum_{t=0}^{T} t \sum_{s=0}^{d} P(t|s) P_3(s)$$

$$= \sum_{s=0}^{d} P_3(s) \sum_{t=0}^{T} tP(t|s) = \sum_{s=0}^{d} P_3(s) E(t|s) \qquad (22)$$

Equation (22) can be simplified. Since the distribution of t for given s is as stated in equation (20), i.e. a hypergeometric distribution, then

$$E(t|s) = \frac{sT}{d} \qquad (23)$$

(See, for example, Reference (23)).

Hence from (22) we find that

$$E(t) = \sum_{s=0}^{d} P_3(s) E(t|s) = \sum_{s=0}^{d} P_3(s) \frac{sT}{d} = \frac{T}{d} E(s) \qquad (24)$$

Thus once we have computed E(s) from (17), we can use the simple relation (24), rather than (19), to compute E(t).

APPENDIX C

TREATMENT OF ATTRITION IN SACLANT THREAT MODEL

This topic is discussed in Appendix 2D of Reference (19). This Appendix attempts to give the essence of the 23-page discussion in the Reference (19) Appendix.

Recall that the distribution of target kills among ships is modelled as a Homogeneous Polya distribution with parameter ß. As discussed in Appendix A of this report, ships are modelled as 'urns', each of initial weight ß, and target kills as 'balls', each of unit weight, which are tossed randomly into the urns in such a way that the chance that a ball falls into a particular urn is

$$\frac{\text{weight of that urn including contents}}{\text{total weight of all urns (including contents)}}$$

In the Reference (19) attrition model, not only 'balls' but also 'lids' are tossed at the urns. If a 'lid' falls on an 'urn', that 'urn' (ship) is killed and cannot receive any more balls (i.e. cannot kill any more targets).

The model still needs a mechanism for deciding whether the next object tossed is a ball or a lid. This is done by introducing a large 'attrition urn' containing balls and lids, from which objects are removed at random. If at a certain stage we still have to toss r balls and b lids at the 'ship urns', it is assumed that the attrition urn then contains rc_r balls and bc_b lids, where c_r and c_b are two integer constants.

An object is drawn at random from the attrition urn, and tossed at the 'ship urns'. Clearly the chance that this object is a ball is

$$rc_r/(rc_r+bc_b)$$

while the chance that the object is a lid is

$$bc_b/(rc_r+bc_b)$$

It is assumed that if the object drawn is a ball, we remove it and (c_r-1) other balls from the attrition urn. Similarly if the object withdrawn is a lid, we remove it and (c_b-1) other lids from the attrition urn. By this means the proportion of 'balls' to 'lids' remaining in the attrition urn is kept at c_r/c_b.

If we define

$$\gamma = c_b/c_r$$

then the chances that the next object drawn is a ball, or a lid, are respectively

$$r/(r+b\gamma) \quad \text{and} \quad b\gamma/(r+b\gamma)$$

Balls are removed randomly from the attrition urn, by the above process, until the attrition urn is empty. If the attrition urn starts with kc_r balls and ac_b lids, when the urn is empty k balls (target kills) and 'a' lids (ship kills) will have been distributed among the n 'ship urns'. One can then write down a hefty probability expression for the distribution of the k 'target kills' and of the a 'ship kills' among the individual ships.

At this stage, without any apparent jusification, Reference (19) assumes that

$$\gamma = \beta$$

where ß is the initial weight of each 'ship urn'.

Once this assumption is made, Reference (19) goes through 14 pages of combinatorial mathematics to prove eventually that the distibution of the 'balls' (target kills) among the ships is Homogeneous Polya with parameter ß. This is the same distribution as would be attained if there were no attrition.

Appendix 2D of Reference (19) therefore concludes that, with the attrition model assumed, the distribution of target kills among ships is still Homogeneous Polya with the same parameter ß as would be found if attrition were ignored.

Comment. Obviously this conclusion depends completely on the assumptions made for the attrition model. Reference (19) admits that if γ is not taken as equal to ß, the distribution of target kills among ships is no longer Homogeneous Polya.

APPENDIX D

INPUTS FOR THE SACLANT THREAT MODEL

INTRODUCTION

This Appendix lists the inputs to program MONTEBF2, with the name and description of each input variable.

LIST OF INPUTS

TITLE The title for this run.

BETA The Polya parameter ß.

NCASES The number of 'years' (i.e. cases) to be run. Each 'year' is run as a completely self-contained calculation.

FYBEG The starting 'year' (e.g. 1996). Results for subsequent 'years' are labelled 1997, 1998, ...

NTT The number of target types.

DEPTH 'Depth of fire'. This means the maximum number of salvos that can be fired against a single target. (If you enter DEPTH = 0, the program assumes that an unlimited number of salvos could be fired against a single target!).

TARGNAMES(I), FOR I = 1 TO NTT The names of the various target types (I).

RNT(I,J) FOR I = 1 TO NTT AND J = 1 TO NCASES The number of targets of each type (I) for each 'year' (J).

NPT The number of shooter types.

PLATNAMES(I) FOR I = 1 TO NPT The names of the various shooter types (I).

NP(I,J) FOR I = 1 TO NPT AND J = 1 TO NCASES The number of shooters of each type (I) for each 'year' (J).

IAP(I) FOR I = 1 TO NPT The initial ordnance load for each shooter type (I).

NRP(I) FOR I = 1 TO NPT The number of rounds in a 'refill chunk' for each shooter type (I).

SALVO(I,J) FOR I = 1 TO NPT AND J = 1 TO NTT The savo size for each shooter type (I) against each target type (J).

PK(I,J) FOR I = 1 TO NPT AND J = 1 TO NTT The single shot kill chance for each shooter type (I) against each target type (J).

The above inputs give us the values of the quantities n, T, IA, R and ß, for use in the Monte Carlo assessment. However, the quantities d and m have to be calculated from other input values

VARIABLES DEDUCED FROM INPUT VARIABLES

From the input values of 'depth of fire' DEPTH, single shot PK(I,J) and salvo size SALVO(I,J), routines AMEAN and MUNI calculate the value of EXWT, the total number of rounds needed to kill all the targets. This is the quantity 'd' needed in the Monte Carlo assessment.

Also from the refill size REPLEN, for any specified depot inventory INV(J), routine INVENTORY calculates RDEP(J), the number of 'ordnance chunks' initially at the depot. This is the quantity 'm' used in the Monte Carlo assessment.

APPENDIX E

CHANCE THAT MAXIMUM INVENTORY IS REQUIRED

Consider the following case for the SACLANT Threat Model (discussed in Reference (19) page 15):

1000 targets; 50 ships; 2 rounds per target kill;
40 rounds initial allowance per ship;
resupply after 20 rounds fired.

Consider the inventory needed to ensure that 100% of the targets are killed. This inventory must be sufficient to cover all distributions of targets among ships, including the extreme case in which one ship engages all 1000 targets. In this case this ship will need 2000 rounds to kill all the targets. The first 40 rounds are in the ship's initial inventory. Therefore the resupply site must pass 2000 - 40 = 1960 rounds to this ship (in batches of 20). Hence the total inventory needed is 1960 plus the 40 x 50 = 2000 rounds initially in the 50 ships, a total of 3960 rounds.

However, this is not the only case in which an inventory of 3960 rounds is required. If a single ship engages 991 of the 1000 targets, that ship will need a total of 2 x 991 = 1982 rounds. 40 of these form the ship's initial inventory, and the remaining 1942 rounds must be obtained from the resupply site. But the resupply site only issues rounds in batches of 20. Hence the site must have on hand 98 batches of 20 rounds, or 1960 rounds. With the 2000 rounds initially allocated to ships, this makes a total inventory of 3960 rounds.

A similar argument shows that a total inventory of 3960 rounds will be needed if a single ship has to enage 992, 993, 994, ... or 999 targets. Hence this inventory will be required whenever one ship has to engage 991 or more of the 1000 targets.

We now work out the chance that such a contingency arises.

CHANCE THAT THE INVENTORY OF 3960 ROUNDS IS NEEDED

Assume that the targets are distributed among the ships in a Homogeneous Polya distribution with parameter ß = 1. Let d be the total number of targets, and n the number of ships. Then it is shown in Reference (19) that the chance that a specified ship gets exactly (d-a) targets is

$$P(a) = \frac{\binom{-\beta}{d-a}\binom{-(n-1)\beta}{a}}{\binom{-n\beta}{d}} \qquad (1)$$

where as usual

$$\binom{n}{r} = \frac{n(n-1)(n-2)\ldots(n-r+1)}{1.2.3.....r} \qquad (2)$$

and this equation holds whether n is positive or negative.

From (2) we see that

$$\binom{-a}{b} = \binom{a+b-1}{b} \qquad (3)$$

If we use (3) in (1) we find that

$$P(a) = \frac{\binom{\beta+d-a-1}{d-a}\binom{(n-1)\beta+a-1}{a}}{\binom{n\beta+d-1}{d}} \qquad (4)$$

When ß =1, d = 1000 and n = 50, equation (4) becomes

$$P(a) = \frac{\binom{1000-a}{1000-a}\binom{48+a}{a}}{\binom{1049}{1000}} = \frac{(48+a)!}{a!48!}\ \frac{49!1000!}{1049!} \qquad (5)$$

This equation was used to calculate, and then sum, the values of P(0), P(1), P(2), ... P(9). The total, which represents the chance that a <u>specified</u> ship would get at least 991 targets, turned out to be

$$1.67447 \times 10^{-75}$$

Hence the chance that <u>any one</u> of the 50 ships engages at least 991 targets is

$$50 \times 1.67447 \times 10^{-75} = 8.3723 \times 10^{-74}$$

REFERENCES

(1) F.H.Fitch, 'Ship Area Air Defence Simulation (SAADS) - Programmer's Guide'. CompEngServ Ltd. November 1992

(2) M.K.Ormrod, F.Fitch and M.A.Gammon, 'Ship Area Air Defence Simulation (SAADS V 4.0) - User's Guide'. March 1993

(3) B.Nguyen, 'Seduction Chaff: A New Defensive Capability for the Ship Area Air Defence Simulation (SAADS)' (Informal Note) 1994

(4) P.Comeau,'Sizing the Fleet of New Shipborne Aircraft', (draft ORAE Project Report)

(5) M.G.Hazen,'A Tactical Development Model for the Use of Anti-Ship Missile Decoys'. MC/ORB Technical Note 6/91. August 1991

(6) Press, Flannery, Teukolsky and Vetterling, 'Numerical Recipes'. Cambridge University Press. 1986

(7) Press, Flannery, Teukolsky and Vetterling, 'Numerical Recipes in C'. Cambridge University Press. 1988

(8) M.S.Craig, 'The Surface Ship Towed Array Study Model Documentation', RANRL Report No.1/86

(9) (no author?) 'TASMAN Model Documentation' May 1994

(10) Michele Fisher, 'The Canadian Active Sonar Variant of the TASMAN Model Release Package'

(11) K.Marse and S.D.Roberts, 'Implementing a Portable FORTRAN Uniform (0,1) Generator' SIMULATION, October 1983

(12) (Frank Fitch?) 'A Portable Random Number Generator for TASMAN', 31 March 1993

(13) Frank Fitch, 'Running TASMAN on VAX'

(14) E.L.Leese, 'The Frequency Distribution of the Sum of Twelve Independent Uniformly Distributed Variables'. May 1995

(15) Fleet Technology Ltd. 'Minefield Clearance Simulation Documentation Set'. December 1991.

(16) Fleet Technology Ltd. 'Parametric Studies of Mine Clearance Effectiveness of Two MCDV-Based Mine Countermeasure Systems Against Enemy Mining of Two Canadian Harbours and Approaches - Final Report'. March 1992. (CONFIDENTIAL)

REFERENCES (CONT.)

(17) LCdr A.G.Hunt, 'Using the Minefield Clearance Simulation (MCS) Program to Study the Effectiveness of Two Maritime Coastal Defence Vessel (MCDV) Mine Countermeasure Systems' ORAE Project Report PR602. July 1992. (CONFIDENTIAL)

(18) E.L.Leese, 'The Frequency Distribution of the Sum of Six Independent Uniformly Distributed Variables'. June 1995

(19) J.J.Boolmarcich, 'Technical Documentation of the Threat Methodology'. QUANTICS Inc., December 1990

(20) F.N.Fritsch and J.Butland, 'A Method for Constructing Local Monotone Piecewise Cubic Interpolants', SIAM Journal of Statistical Computation, Vol 5, No. 2, June 1984

(21) Memo QM 61:95 from J.J. Bolmarcich of QUANTICS Inc. to DMOR, dated 31 July 1995

(22) Memo 3553-1 (DMOR) from DMOR to J.J. Bolmarcich of QUANTICS dated 2 February 1996

(23) Walpole and Myers, 'Probability and Statistics for Engineers and Scientists', 1972, pages 88 to 92.

(24) Memo QL 35:96 from J.J. Bolmarcich of QUANTICS Inc. to DMOR, dated 16 May 1996

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

DOMUS SOFTWARE LTD
309 COOPER STREET
OTTAWA, ONTARIO

2. SECURITY CLASSIFICATION (overall security classification of the document. including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

A REVIEW OF SOME DMOR COMPUTER SIMULATIONS (U)

4. AUTHORS (Last name, first name, middle initial)

LEESE, ERIC L.

5. DATE OF PUBLICATION (month and year of publication of document)

JUNE 1996

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

84

6b. NO. OF REFS (total cited in document)

24

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

TECHNICAL REPORT. OCTOBER 1994 TO JUNE 1996

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

MORT/ORD

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

ORA CONTRACT REQUEST NO: W0141-94-S0-12

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

MORT CONTRACTOR REPORT CR9601

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(XX) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availabilty (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

This Contractor Report examines a number of DMOR computer simulations, already written. Each simulation was tested and, if necessary, corrected to ensure that it worked correctly for various cases of practical importance. Where possible, input variables whose values were "hard-wired"into the source program were replaced by values that could be assigned at the input stage. Other amendments were made to reduce the execution time of the programs. In one case (ENDLESS), the source code had to be translated from a langugage suitable for the Macintosh computer to a language that could be compiled by the Microsoft IBM PC software.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

OPERATIONAL RESEARCH
MARITIME
SIMULATION
AIR DEFENCE
DECOYS
BARRIER PATROL
LOGISTICS
COMBINATORICS
PROBABILITY

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM